7-14-1996

Space adventure

David Seah

ROCHESTER INSTITUTE OF TECHNOLOGY

A Thesis Submitted to the Faculty of

The College of Imaging Arts and Sciences

In Candidacy for the Degree of

MASTER OF FINE ARTS

SPACE ADVENTURE

by

David Ingram Seah

July 14, 1996

APPROVAL

CHIEF ADVISOR	**ROBERT KEOUGH**	_____ SIGNATURE	9-10-96 DATE
ASSOCIATE ADVISOR	**JAMES VERHAGUE**	_____ SIGNATURE	9.10.96 DATE
ASSOCIATE ADVISOR	**HEINZ KLINKON**	_____ SIGNATURE	9/10/96 DATE
DEPARTMENT CHAIRPERSON	**MARY ANNE BEGLAND**	_____ SIGNATURE	9.12.96 DATE

I, **David Ingram Seah**, hereby grant permission to the Wallace Memorial Library of RIT to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

_____ SIGNATURE, 9.2.96 DATE

Dedicated to the memory of my mother,

Grace H. Seah
1938–1992

...who probably would have gotten
a big kick out of my career shift...

CONTENTS

INTRODUCTION

The idea of making a multiplayer game draws its inspiration from fictional and historic sources. As a first step, I identified a number of key influences that would help guide the development of the project.

Inspirations for Thesis

fictional inspirations

The initial inspiration was the film *Porco Rosso*, an animated film directed by Hayao Miyazaki. A notable element of Miyazaki's work has always been the stylish animation of flight. Through his films, I gained a renewed appreciation of early aviation history, which could be characterized with open-air cockpits and "seat-of-the-pants" flight. In this period, the men and women who mastered their fragile craft were the darlings of the media; they were symbols of courage and independence in a forward-thinking, technology-driven society. I wanted to capture this sense of adventure in my project by building an environment that would encourage glorious risk taking.

historical influences

Rather than recreate that particular time period, though, I chose to call upon a childhood interest: space travel. Every kid I knew in the fourth grade became intensely interested in spaceships when *Star Wars* was released in 1976. *Star Wars* brought dynamic spacecraft to the screen. The spaceships weren't 1930s pulp science fiction; nor were they based on NASA vehicle projections. They were unique, and they shared the spotlight equally with the characters of the movie. Why not, then, project the spirit of 1920s into the era of space travel?

Although *Star Wars* was my first inspiration as far as spaceships are concerned, I didn't want to just duplicate it. I saw more similarities between 1930s aviation—with its relatively stable but still-shaky industrial base—to Man's first steps towards the commercialization of Space. In this projection, spacecraft would be fragile and limited in range. Spaceport facilities would be few and far apart. Navigation would be an undeveloped science. The men and women who would live in this environment wouldn't be part of a developed commercial marketplace. They would be explorers and frontiers people. They would take risks and seek to distinguish themselves with individual accomplishment, and above all be there for the experience of it all.

multiplayer environments

The building of a new society in space is the effort of many people. Had I wanted to limit the experience to a single explorer, I could have chosen to implement a standard graphical adventure-game (a.k.a. *MYST*). However, for individuals to meaningfully distinguish themselves, there must be *other individuals*. Rather than have some abstract accomplishment, I wanted to implement a social experience using computer technology—a *multiplayer* environment. Given the same presentation quality, a multiplayer game is almost always more fun than a single-player game. Such a game can be more meaningful to the participating players with even crude graphics and sound. I wanted to build a *shared environment*, not just some abstract game. Games have objectives that are solved. An environment becomes a playground, where people meet to participate in some kind of activity or game of their own making. I envisioned players in the game making their own headlines in a future, historically-significant era, forging a new age of independence in Space. This vision was inspired by my encounter with *Chronicle of Aviation*, a history of flight presented in a contemporary newspaper style. The material made me aware of how much the entire field of aviation is made of the daring contributions, successes and failures of many individuals around the world.

Design Criteria

From the above inspirations, I defined two criteria by which I could subjectively measure the "success" of my project: **Implement a multiplayer game** and **Evoke a sense of the 1920s aviation history in the context of Space Exploration.** This were broadly-defined, and seemed quite achievable at the time. The design of the game is described in the next section.

subconscious goals

There were also several unstated criteria that are baggage from my previous field of study. I was not aware of their influence on my design activities until late in the project. Briefly, these criteria are:

Implement an architecturally-clean, expandable, and elegant computer project

As a rusty programmer and computer engineer, I have certain programming aesthetics that I try to adhere to. An *architecturally-clean* project is one that can be broken down into a well-defined system of interacting parts. There are no unnecessary redundancies or "patches". An *expandable* project is one that can grow without too much additional programming. It is designed from the beginning to scale upwards in size (e.g.: more graphics, more sound, etc.) without drastic modification. *Elegant* projects are one that are especially clever in their implementation while retaining some semblance of simplicity and efficiency in coding. Achieving all three of these criteria is always an unspoken priority.

Implement an elegant yet sophisticated role playing system

An aspect of a complete multiplayer environment is the role playing system. Role playing games (RPGs), in the context of products like *Dungeons & Dragons*, consist of a set of rules that detail how a player may act within an imagined environment. There are rules that govern character development, the exchange of money, social activities, skills, and knowledge within the context of the "play world". The goal of an RPG designer is to balance rule complexity with playability. If the rules are too few to adequately address a player's action within the game world, the result is chaos. If the rules are too complex, the game degenerates into a statistical nightmare, with little entertainment value. RPG design has been something of a fascination for me for the past few years, so I thought it would be interesting to try and design my own.

I attempted to design a simple RPG system that embodied the traits of a good software engineering project: clean design, expandable, and elegant. In retrospect, I should have just borrowed an existing system and implemented that quickly. The final project, therefore, does not implement a functioning RPG system. Some support code is present, but it is essentially vestigial.

Custom X-Object Extensions and Lingo Programming

As I've worked more with Macromedia Director, I've come to recognize its strengths in rapid prototyping. However, there were times that I wanted to enhance the capabilities of the program, either to improve performance or just to "do something cool." The urge to do some coding was something that surfaced early in the project.

There were several components to the thesis that would have benefited from custom programming. The role playing system, for example, would be easier to write in a computer language like C. Additional performance-boosting

considerations, such as the need to generate many more sprites than Director could handle, also fell under the custom programming task umbrella.

I spent time reading and researching Macintosh Toolbox programming, building tools and understanding for the time when I would actually apply this knowledge to my project.

I coded some special tools built to make the design process easier. Conceptually, the idea of a tile-based visual display was a clean one, as it allows one to create relatively large spaces with few parts with one system. This offered immense design possibilities, but the prospect of creating such a space by hand was initially quite daunting. I evolved a procedure and set of custom X-Objects to make this task simple.

Authenticity, Consistency, and Correctness

Another factor that weighed heavily on my creative process was my subconscious need to "properly research" a subject before creating it. This tendency may stem from an innate sense of caution, but it also reflects a desire to be well-read on those topics I find interesting.

One aspect that hampered my creative process was my interest in discovering the difference between "authentic-seeming" and "fake" environments. At the time, I thought that "attention to detail" was the key, but I also was searching for the "force" that bound these components together. This quest may be a manifestation of my engineering experience. I typically know the scientific principles of a computer design before I attempt one. In an artistic venue, this isn't necessarily the case. "I'll cross that bridge when I reach it" would have been a good maxim to keep in mind during the design phase.

accomplishments

Although my visual creative process was thus subverted, I did succeed in creating a two-player, exploratory environment, complete with rudimentary networking support. I created useful world-construction tools that would have allowed me to create an extensive world. I learned the fundamentals of advanced Macintosh programming, and developed two custom X-Object extensions to the Macromedia Director environment. I was able to integrate the two-dimensional world with an interface that, while not reminiscent of the 1920s, did encompass the scope of interactivity I had planned. I researched the space environment to the point where my concepts were compatible with science. Finally, the images and animation that I did produce met with a level of quality that I found acceptable.

What Next?

The concept of my thesis was relatively simple, but supporting all these elements in an attempt to build a believable, *interactive* environment was a formidable task. I began the project by evaluating my initial inspirations, researching Space, and defining some goals. The implementation is discussed in the next section.

RESEARCH

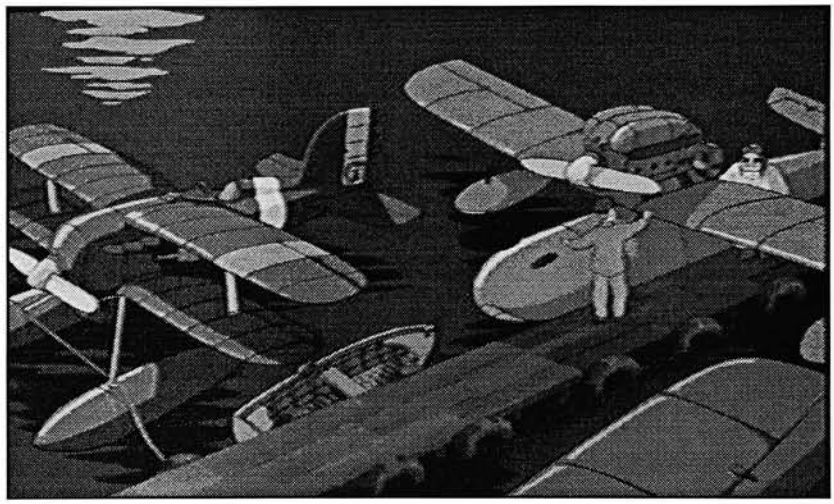
The design of this project was guided by many books, movies, and software titles. Raw inspiration came from the fictional and historical works, while specific design decisions were guided by general science research and computer texts.

Fictional Sources

As mentioned in the introduction, my initial interest in the 1920s was inspired by the Japanese animated film *Porco Rosso (The Crimson Pig).* This story takes place in the Mediterranean, a few years after the First World War. The hero, Marco, is an veteran Italian pilot who now makes his living as an "air pirate bounty hunter." The film's creator and director, Hayao Miyazaki, devotes considerable attention to the aircraft. Marco's plane, for example, was a bright red Italian custom seaplane, with an over-the-wing engine. It appears to be one of the last biplane designs. Marco's rival, the American pilot Curtis, flies a blue Curtiss Seaplane Racer, one of the first monococque, single internally-braced wing designs. Both designs are distinctive, open cockpit, hands-on-stick fliers that could be flown right up to a seaside cafe.

Figure 1: *Porco Rosso*

Still from Hayao Miyazaki's 1991 animated feature film.

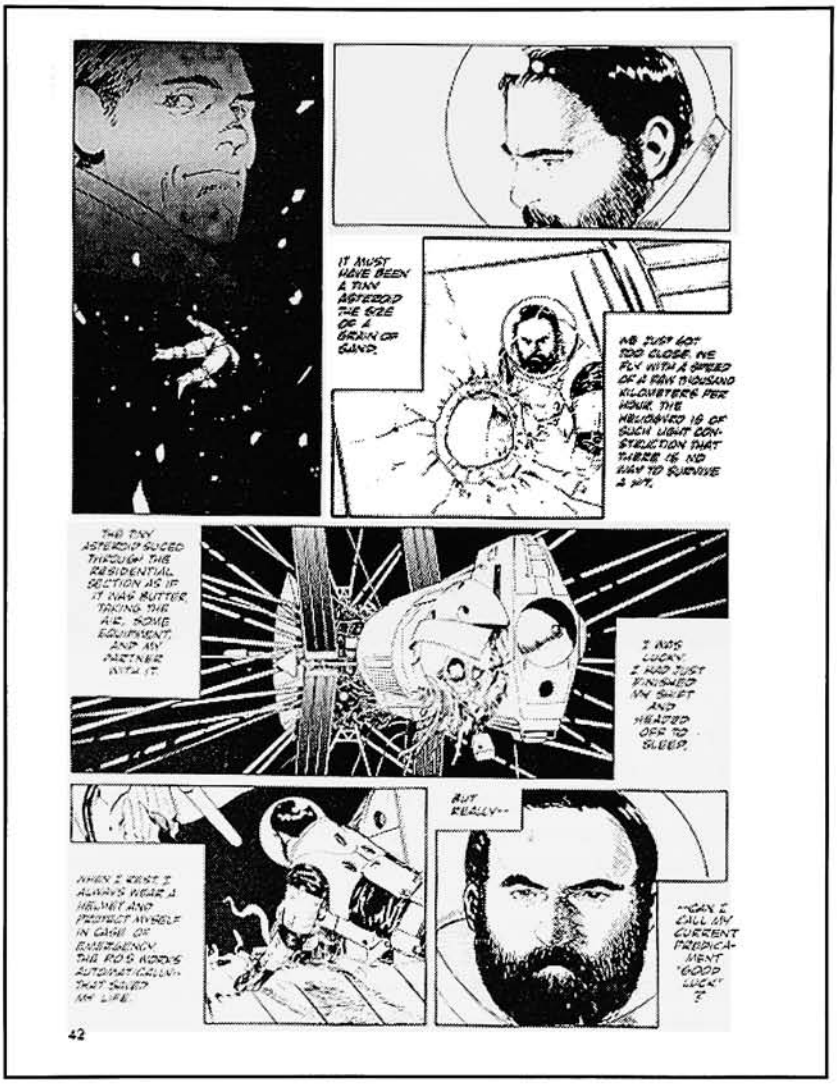


I also found inspiration in Yukinobu Hoshino's graphic novel *2001 Nights.* In this work, people were bound together in a common destiny, working together to achieve Man's transition to the final frontier. Hoshino's work is composed of a series of vignettes, each a poignant story about people and the dangers of space exploration. Many of the stories were bittersweet in their conclusion, but this gave me motivation to develop a multiplayer game environment. By providing communication and role-playing features, real people could engage in exploration in a real social context, leading to shared experiences of an epic nature. It was my hope that a suitably-detailed environment would promote imagination, and allow the creation of stories that reflected all three fictional influences.

Upon mention by Heinz Klinkon, I read Yves St. Exuberry's book *Wind, Rain, Sand, Stars.* This book appears to be the memoirs of a French Air Post pilot in North Africa, describing the dangers of flying in the early 1900s, when aircraft were first being recruited for commercial uses. Air Post pilots

Figure 2: *2001 Nights*

A page from Yukinobu Hoshino's award-winning graphic novel. This page shows the english adaptation of the original japanese.



were the best paid in the world, betting their lives on uncertain engine performance, poor weather, and hostile natives. Some of these elements worked their way into a brief short story I wrote as a warm-up exercise (see Appendix C for the text).

Design Precedents

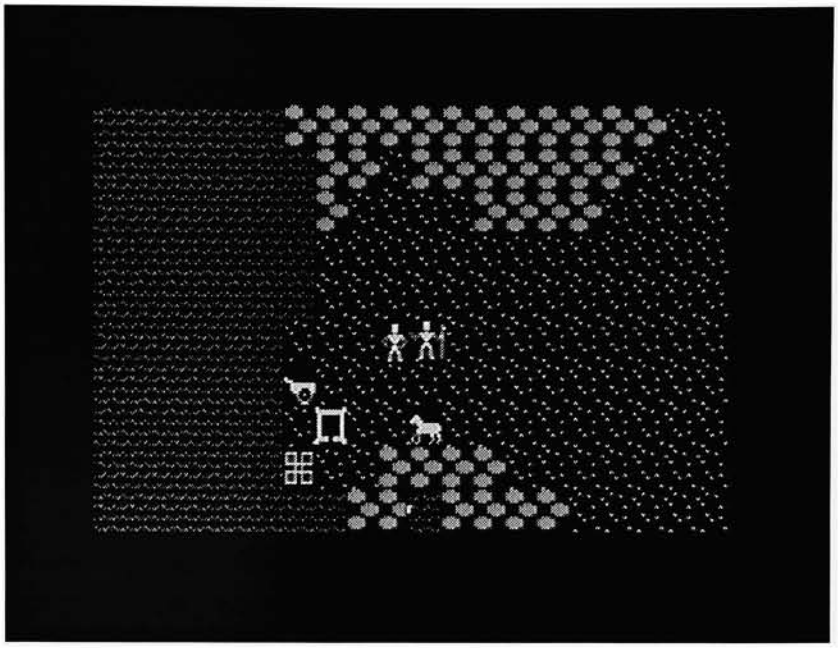
I was influenced by several kinds of software, ranging from 1980s computer games to network-based communications tools.

games

Ultima: This is the grandfather of 2D, top-view, tile-based role-playing games. This game was introduced in 1981, and dominated computer role-playing for a number of years. It has fallen out of favor in recent years, primarily due to the introduction of 3D games. The original game is written in Applesoft BASIC with assembly language extensions. This is not unlike the Director Lingo / XObject I had proposed.

Figure 3: *Ultima*

Richard Garriot's original bitmap-graphics role playing game. Successive versions added more interface features.



user interface style

Direct Manipulation Interfaces: 3D role playing games present the player with a perspective view of monster-infested dungeons. The player interacts with the dungeon by pressing keys on the keyboard. With the widespread introduction of the computer mouse in 1985, games started to move away from the keyboard. The first game to use the mouse as the primary game interface was FTL's *Dungeon Master* for the Atari 520ST. Instead of pressing a key to "get" an item, you could actually "reach in" to the game and "drag" the item to a virtual knapsack. The first Ultima games, written before the advent of computer mice, did not have this kind of interface. I decided to allow the user to click directly on the tile world to interact with it, rather than provide button commands.

internet tools

IRC: The Internet has spawned a communication tool known as *Internet Relay Chat (IRC)*. In IRC, you can talk to people in near real-time, in "channels" that are not unlike CB-radio. Like CB, participants in IRC often choose "handles", which are used in the place of one's real name. IRC also makes provision for "acting out" actions as a kind of running commentary. For example, a typical IRC session might look like this:

```
<BugKiller> Hey, there's a spider on the keyboard.
<Joe> Squash it!
* Fred lifts his hand and smites the spider with a mighty
blow!
<BugKiller> Take that!
* Joe feels sick.
```

The text preceded by a <*> are the actions. I decided to provide this basic feature in my project. The communication model is essentially that of IRC. You type into a text window, and this text is transmitted to other players on the network.

MUDs: The Internet also has something called a *Multi-User Dungeon (MUD).* This is a text-based creation that may have been spawned from IRC. Creative individuals on IRC can "tell a story" by acting out the drama. However, there is nothing to enforce compliance to the laws of gravity, and chaos soon dominates the action. When someone can just say, "I beat up the dragon and take all the gold," the gaming possibilities are quite limited. A MUD is a program that prevents this by enforcing the rules. The program will also keep track of monsters, money, combat rules, and so forth. MUD programming can be quite sophisticated.

Space Science

I read several books on Space Science, with topics ranging from the effect of radiation on materials to the geology of the Moon. I needed to establish several environmental issues and a plausible motivation for man to go out into space in the first place. The observations that seemed to be useful in constructing the background world were assembled into a set of notes.

colonization

The book *Out of the Cradle: Exploring the Frontiers Beyond Earth* introduced the economic and social motivations of space exploration. It briefly described and illustrated the colonization of the Moon, Mars, and planets beyond without getting into too much detail. For more technical information, *The Starflight Handbook: a Pioneer's Guide to Interstellar Travel* discussed the principles of space flight and propulsion systems. These two sources seemed to be compatible with my thoughts regarding "the 1920s in Space." The 1920s aviation technology was fragile, and exploration was promoted by both economic and adventure-seeking concerns. Space is a dangerous place, but this is the backdrop for achieving "fortune and glory." I decided to restrict activity to the vicinity of the Earth and the Moon, with Mars and the Lagrange Points (stable areas in the lunar orbit) as future destinations.

environment

For information on the space environment, *Space Stations and Platforms* provided a wealth of detail. Facts about Low Earth Orbit, the Van Allen Radiation Belts, the effects of vacuum on the human body, solar activity, and space construction techniques were all gleaned from this source and compiled into a fact sheet (see Appendix B). A less scientific source was *Deep Space*, a *Cyberpunk* role-playing game supplement. In this sourcebook, the equations presented in *Space Stations and Platforms* were simplified and translated into terms of economic cost. Some information, such as the location of all Lagrangian points and the description of "deltaV" (the energy cost of launching a rocket in a gravity well) were described in this source.

I made the decision to consciously avoid using the *Cyberpunk* metaphor for role-playing. This genre, while fascinating, is at odds with the "nostalgic 1920s" design I had in mind. *Cyberpunk* is all about advanced technology in a disintegrating society. I was more interested in building a society.

Space Vehicle Design

I chose to design spacecraft that were relatively unstreamlined and experimental. This was intended to reflect the kind of styling (or lack thereof) associated with Man's first commercial steps into Space. The aforementioned book *Space Stations and Platforms* greatly influenced the design of my spacecraft. From this (and to a lesser extent *Deep Space*), I derived a class of vehicles that fit into the operational profile of a space-based transportation system. The basic vehicles types are:

space plane — A reusable, high-speed transatmospheric craft (HST) that could take off from Earth and deliver its payload into low earth orbit (LEO). It can return to the Earth with another payload. This is similar to the US Space Shuttle in concept.

space station — A manned space station in LEO, which serves as a transfer point to other space-based vehicles. Such stations would have both commercial and scientific value. They would be constructed using modular construction techniques pioneered by NASA and the ESA. Reliance on prefabricated, standardized components would make space construction activity less expensive.

intra-orbital transport — The OMV, used for construction and utility work in LEO, can also travel to other destinations in the same orbit. These are short-range vehicles, intended for maintenance work.

spacesuit — Extra-Vehicular Suits (EVS) are the "spacesuits" worn during spacewalks, when an OMV is not versatile enough.

orbital transport — An orbital transfer vehicle (OTV), which is used to transfer to and from higher orbits (to geosynchronous or lunar orbit). These are larger vehicles, with enough fuel capacity for a round trip.

lunar station — A Lunar Station in orbit over the Moon itself, used for housing workers on the moon prior to the establishment of a lunar base. Also serves as a monitoring station and OTV to Surface transfer point.

lunar lander — A Lunar Soft Lander. This is like the Apollo Lunar Excursion Module (LEM) with greater reusability and cargo capacity.

lunar base — A Lunar Base, established for scientific and commercial research. The base is large, capable of sustaining several hundred permanent residents.

lunar rovers — Lunar Transports, in the form of "hoppers" (short range rockets) and "rovers" (wheeled vehicles) on the surface.

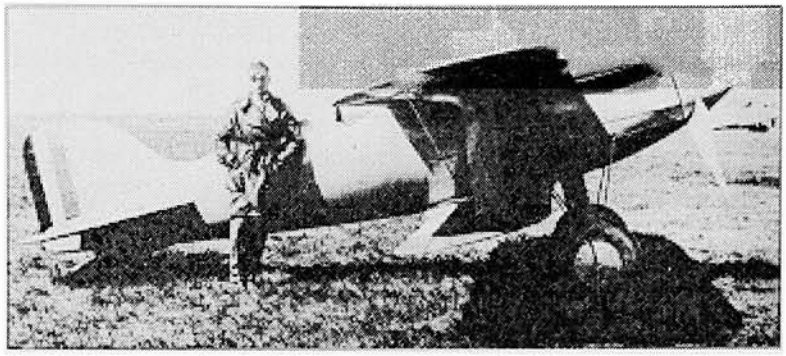
Vehicle Design Philosophy

The vehicles described above were defined for a number of multi-purpose roles. For example, The establishment of a permanent, low earth orbit space station requires modular construction techniques using such vehicles (particularly OMVs and EVSs). The vehicles themselves would be built out of the same modular technology when possible. Extruded beams used in station construction are easily adaptable to building OTVs, with a few extra modules, zero-G metallurgy, and rocket modules. Use of these common components also leads to a recognizable "space style."

The modular nature of these craft also lend themselves easily to customization, which introduces an interesting activity for space colonists. As in the 1920s, when aviation was as much luck as it was engineering, the design and building of "homebuilts" could evolve into competitive racing, exploration, and feats of heroic exploration.

Figure 4: Photos from *Chronicles of Aviation*

Fragile aircraft technology of the 1920s.



Curtiss built racers for the US Army as well as the US Navy. The clean lines of the 266-mph R-8 (originally the Navy's R2C-1) are well displayed here.

Appearing in 1923, the Curtiss PW-8 made the first dawn-to-dusk crossing of the US on June 23, 1924. Illustrated is the XPW-8 prototype 23-1201.

Programming Resources

X-Object Development: Role-playing games require many rules and lots of data. Information must be stored regarding the effectiveness of weapons, the capabilities and possessions of individual characters, locations within the game, game descriptions, and so on. Director's scripting language is not well-suited to this kind of programming. A language like C, on the other hand, is quite adept at it. To provide database handling features, I purchased Macromedia's *X-Object Development Kit*, which allows a programmer to extend the capabilities of Director using C.

Macintosh Toolbox Programming: A second hurdle in the way of implementing this game in Director was speed. While version 4.0 featured improved speed, the general graphics handling capabilities of Director were too limited to implement an efficient 2D tile display. Using traditional Director techniques would not work. Using the *X*-Object developer kit, I was able to improve the speed of the tile display by several times. This required study of the Macintosh computer as a programming environment.

IMPLEMENTATION

Having decided on the creative direction of the project, I defined several technical requirements, borrowing heavily from existing computer metaphors for gaming and communication systems.

Overview

Desirable Elements

Multiple Player: More than one person should be able to participate in the game at one time. Multiple players make the game experience into a social activity.

Real-Time: The sense of time in the game should be roughly equivalent to our own sense of time in the real world. Many non-computer games use a "step-based" or "turn-based" model, in which each player can declare their "actions" for a turn. After the actions have been declared, the game program figures out the outcomes of each action. For example, Player A may say, "For the next turn, I will try and shoot Player B." Player B may say, "For the next turn, I will duck behind a tree". The outcome (often determined by a statistical table or process involving dice) is declared. Although this makes for neat record keeping, it is less than interactive. For a Real-Time game, your actions in the game happen whenever you perform them.Since the computer can automate the rules, the game's pace can be accelerated.

Shared Environment: All players would participate in the same virtual space. Each player can "see" the other player if they are within visual range. Items that are manipulated by each player may affect other players. The computer should maintain a record of the state of the environment so it is presented to each player in the same way.

Multiple Station: Instead of having several players share the same computer, several computers would be used to access a shared environment. This requires some form of computer networking. In this project, I began to implement a LocalTalk-based network scheme, but abandoned this in favor of the technically simpler "direct serial" connection. The disadvantage of this scheme is that only two computers could be linked at one time, but for the purposes of demonstration I felt this was adequate.

Technical Decisions

There were two main technical constraints on this project:

2D Graphics Display: As I do not possess the knowledge to implement a real-time, three-dimensional virtual world, I settled for a two-dimensional, tile-based display. A "tile-based" display shows images that are made up of smaller pieces. As in the board game *Scrabble*, which makes crosswords out of individual letter tiles, a tile-based image is made up of smaller pictures. This graphics format has ample precedence in the computer role-playing game world. Richard Garriott's *Ultima* series is the quintessential example of this type of display.

Macromedia Director: I chose *Director* to be my development platform. Previous work I had done using "trails" with *Lingo* programming made it possible to implement a 2D tile-based system. Director is also extensible through the use of *X-Object* programming in the C computer language, which could be used to provide database functions that are not directly supported by Lingo. This flexibility has an advantage over a traditional pro-

gramming language like C, which would have required many more man-weeks of study. The biggest disadvantage of Director is its speed. A program written in C runs many times faster than the equivalent Director movie. However, I was able to extend Director with some custom C programming to provide the necessary speed.

Interface Design Decisions

With much of the preliminary background and technical research out of the way, the design of the interface took the form of a multi-window display. The design incorporates a 2D overview window, a character status window, an animation window, a console window, and a control object (see next page for a diagram.) The final project would run solely on the Macintosh, because it is a more stable multimedia platform than the PC. Additionally, I had access to more than one Mac for multiplayer testing.

Software Decisions

I chose to use Macromedia Director 4.0 for its powerful tools, extensibility, and flexible scripting language. For image production, however, I used both the Mac and PC as needed. Most initial design was done with Illustrator 5.5 on the Mac, but the PC was used for both Photoshop and animation work for convenience and speed.

The animations and still images of the spacecraft were created entirely with Autodesk *3D Studio R4* (hereafter referred to as simply *3DS.*) The designs themselves were based on sketches I made, using the construction concepts described in the Introduction. These craft designs fit into the NASA vehicle concepts, following modular design principles. As the players explore, the short animations depict these craft traveling from location to location.

Hardware Decisions

The final game as shown at the thesis show involved two PowerMac 7100 computers, linked via serial cable. Each Mac ran one copy of the game project, keeping each other updated via the serial cable. As one player moved, both computer displays were updated. Players could talk to each other via a Console window.

Thesis Show Decisions

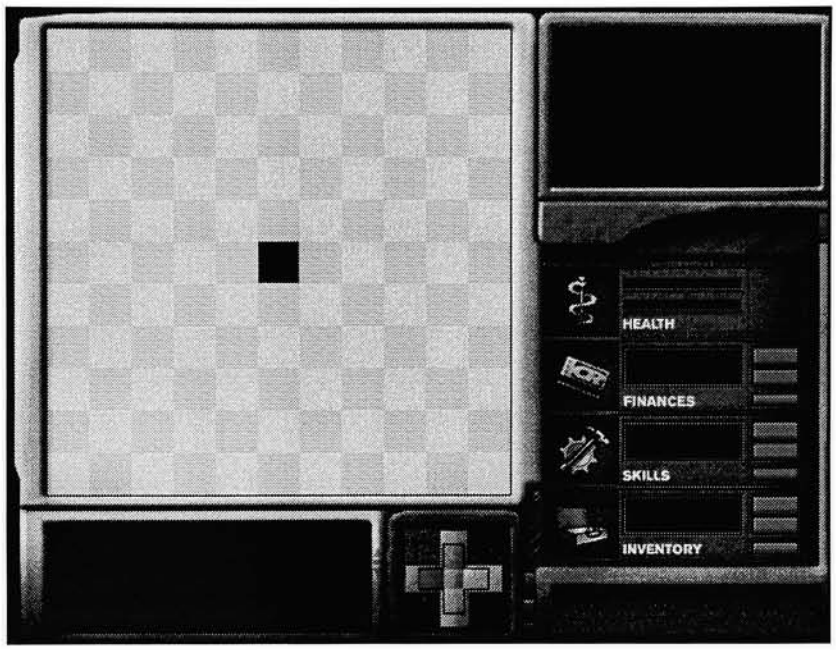
For the thesis show presentation, the design of the game was kept very simple. The players could talk to each other while walking around a rather sparse set of maps presented on the 2D overview window. The simple objective was to "find the orbital transfer vehicle" and fly to the moon. This required cooperation on the part of both players, as the orbital transfer vehicle requires two pilots onboard. The character status window was nonfunctional, but the Animation window depicted vehicles in flight as your character boarded them.

Most of the exploration takes place in the 2D Overview Window. The world can consist of several maps that are loaded and displayed in this window. Each map can be larger than the screen, and can contain many different tiles.

INTERFACE DESIGN

The playing screen provides several distinct "interfaces" to the user. The player interacts directly with the game world via the 2D Overhead and Console windows. These two elements are positioned on the left side of the screen, and are drawn with white borders. For additional detail, the Animation and Character Status windows are provided on the right. These are drawn in a darker metallic brown to distinguish them from the main interactive screen elements.

Figure 5: Interface Design



separation of function

The separation of these two functions is somewhat arbitrary. My reasoning was that the most-used elements should be grouped together. The player's actions largely take place inside the main 2D Overhead window. A quick drop of the eye's focus to the Console window allows the player to check the text status easily. If the player needs to investigate some personal character stats, then attention can be turned to the separate elements on the right.

This might have worked well if one design flaw had been corrected: the interactive Directional Widget, which controls player movement within the game. Players have to visually find and click the widget, which *takes the attention away from all of the other interface elements*. Addition of keyboard control keys would have rectified this situation. The current design is just too distracting.

graphics construction

The graphics for the interface were created with a variety of tools. Layout of black & white outlines was done with Illustrator 5.5 and CorelDraw 5.0. The screen is divided into a 3x3, equally-proportioned grid. The active display areas of each control window are positioned carefully to fall on a 4-pixel boundary, which involved some fudging from the original grid. The reason for this alignment is that QuickDraw can more quickly redraw graphics that are positioned at the beginning of a 32-bit (4-byte) memory block, due to the way microprocessors interface to computer memory subsystems. Although the speed gain can be very slight, I needed as much speed as I could muster to have reasonable screen update times.

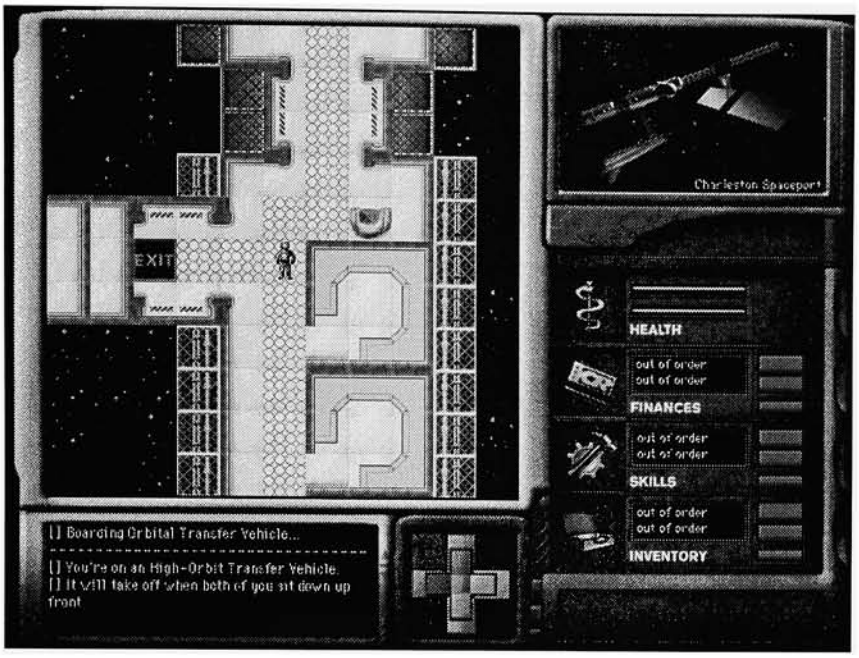


Figure 6: Interface

Screen shot of game in progress.

Clockwise from upper left:

The **2D Overview Window** is the "main view" of the environment, presented in a flat schematic style. The world is much larger than this screen, so the environment scrolls around as you move. The player, represented by the spacesuit icon, is always centered in the screen.

The **Animation Window** augments the 2D tile display with more fully rendered graphics. The 2D schematic style is rather dull, so the animation window shows the environment in a more illustrative style. The hope was that "a sense of place" would be established in the player's mind. This window can contain useful text information such as the name of a place. Text is overlaid the graphics, which can be either a still or an animation.

The **Character Status Window** is where the player can examine his/her character statistics. Game elements such as health, wealth, skills, and material possessions would be interacted with here.

The **System Control Window** contains controls for computer system parameters such as sound volume. In the thesis presentation, it was used to hide the "restart game" button.

The **Directional Widget** is used to move around the world. It works as a direction diamond; clicking on top of the cross shape will move the player's character toward the top.

The **Console Window** is used for talking to other players in the game, similar to the IRC session described previously. There are also a number of special commands, which are distinguished from conversation by preceding the sentence with a "/" (slash) character.

Some icons for the Character Status window were created from the *CorelDraw* 5.0 clip art collection. They were converted from a vector format to a Photoshop image. Subsequent coloring was done entirely within Photoshop 3.0. To achieve the slight metallic look of the brown windows, the "dodge/burn" tools were used to created diagonal striations in a large brown square. Hue variations were painted in using the "airbrush hue mode". Photoshop "curves" and "levels" were used to pop out the metallic highlights even more. Gaussian noise was added to create a slightly-textured finish to the entire surface.

tile-based display

The 2D Overhead Window contains 11 rows of 11 tiles apiece (11x11 tiles in size.) Each tile is 32x32 pixels, chosen because these happened to be the size of the standard Macintosh Icon. I had originally planned to use a standard Icon Editor for the Mac to make my tiles before I decided to create the TileGrabber utility (described in a later part of this document.)

MAP EDITOR

The thought of making 2D map structures involving hundreds of tiles was quite depressing. I originally had planned to "fake" tiles by using very large PICT files, but the memory requirements make this impractical. Plotting basic terrain types on graph paper proved to be tedious work. At this point, I decided to create a Map Editor using Director. This approach solved several problems. First of all, it would allow me to create large maps interactively, making the creative process much easier. Second, to build a Map Editor would require designing the final "map display" code for the game itself. If designed properly, I could easily "copy and paste" large portions of my code between the Map Editor and the final game.

Functionality

The Map Editor builds large 2D maps out of smaller tiles using a point-and-click interface. The user can place tiles on a map grid of arbitrary size. Additionally, the Map Editor will let you place ITEMS and ENVIRONMENT codes on the map grid. These codes are stored separately from the "basic background" that the 2D map provides, allowing the map designer to place "specials" and "moveable items". The ENVIRONMENT codes allowed me to specify where the player's character may physically go, provide tags for "dangerous radiation zones," and so forth.

practical limits

The map can be saved to a standard text file. Although there is no hardcoded limit to the size of the map, the game becomes rather sluggish beyond 40x40 (1600) tiles. The text file is created with a text editor. *BBEdit*, a text editor from Bare Bones Software, is the best such program I could find. The map can also be loaded from the file as well, so the designer can resume work without recreating the whole map.

The Map Editor can display 15x15 (225) tiles on the screen simultaneously. The map can be scrolled with the keyboard arrow keys if it is larger than 15x15 tiles. Other convenience features are undo, "pick from screen" for tile selection, and numeric-keypad tile number entry.

Map Data Structures

The map data is stored as a two-dimensional Lingo **list**. Lists are new in Director 4.0; they are similar to arrays in BASIC or C. You can store a list that consist of more lists, which is what I do to build my two-dimensional map data structure. The position of a number within the list corresponds directly to one on the map. It is like numbering a piece of graph paper both horizontally and vertically.

There are three 2D list structures: TILE, ITEM, and ENVIRONMENT. Each 2D list stores a number for a different aspect of the map. The TILE list stores the tile number that corresponds to a particular 32x32 pixel graphic. The Map Editor redraws the tiles by looking at this number, selecting the tile that goes with it, and putting it on the screen (more on the exact technique later). The majority of images in the 2D Overhead Window consist of these tiles. The ITEM list stores the location of special items on the map, which are drawn on top of the TILEs. The ENVIRONMENT list stores special information about a particular location on the map. One code means "the player can not step on this tile"...this is the NP (no pass) code. Other codes may mean different things, such as "hard vacuum—if the player has no spacesuit, he will die." In the version for this thesis, I implemented only the NP code.

Each list contains the same number of tiles as the map. For a 10 horizontal by 20 vertical (10x20) tile map, each list contains 200 tiles.

storing map data

As mentioned above, the map is stored as a regular text file, containing a number of carriage-return delimited lines. The first two lines respectively contain the horizontal (X) and vertical (Y) size of the map, followed by the tile data itself. Each line represents a row of tiles, items, or environment codes on the 2D map. This data is expressed as a bracketed list of X numbers (conforming to the Lingo LIST syntax) that are Y lines long. Since there are three kinds of lists, the text file must contain three rows of numbers for each row on the map, there are 3*Y lines of numbers in the text file.

Fast Scrolling Tiles

Although Director 4.0 has provision for up to 48 sprites on the screen, this is still too few to represent an 11x11 grid of tiles (121 tiles). I make use of Director "puppets" and the "trails" ink property to simulate more sprites. The basic technique is to use one sprite channel (I used channel 48) to act as an "ink stamp." With trails active, Director's drawing engine does not erase the sprite as it moves around, leaving a "trail" of pixels. By controlling where that sprite is drawn while varying the cast member associated with it, an entire screen of tiles can be painted. The steps are:

1. Puppet the ink sprite
2. Starting from the upper left corner, look up the tile that should be displayed from the TILE list. Set the castnum of the ink sprite to this.
3. Move the ink sprite to this location and update the stage so it is drawn.
4. Look up the tile that should be stamped on the next position. Again, get this number from the TILE list.
5. Repeat 3 until the whole tile display is drawn.

Since the map is larger than the screen display, some calculations must be made to determine which portions of the TILE list should be used in updat-

ing the screen. This is more fully documented in the source code (see Appendix A).

While this technique allows one to simulate having 121 sprites with just 1 sprite channel, it has the drawback of being somewhat slow. On a PowerMac 7100/66AV, it takes approximately half a second to draw all 121 tiles. For movement purposes, this is very sluggish. This is also bad for animation, as the redraw is slow enough to be seen. This imparts an "undulating" motion to the screen updates ("screen flap"), which looks awful.

XObject graphics extensions

To improve speed and eliminate the flapping effect, I wrote a very simple XObject extension with Symantec C 7.0 and the Macromedia XObject Development Kit. The strategy is simple: Draw fewer tiles in Lingo. If I could draw 10 times fewer tiles with my Lingo code, the display will be 10 times faster. In a scrolling field of tiles, we can take advantage of tile redundancy. If the screen scrolls up, the top row of tiles disappears, and a new row of tiles appears at the bottom. The remaining rows of tiles are unchanged except in position. A very fast position change of graphic images can be accomplished with the ScrollRect QuickDraw call. which copies memory from one part of the screen to another. Since the graphic images on a Macintosh screen are represented by memory, moving that memory moves the images. A huge speed increase can be realized by this technique. The only portion of the screen that has to be redrawn is a single row of tiles.

Most of the difficulty in getting the XObject extension to work was learning how to use Symantec C and the Macintosh Toolbox. In addition, the Mac Toolbox was original designed for Pascal, another programming language. Since I was using C, I had to massage data structures into one form or another to get the code to work. Another factor was my lack of familiarity with these data structures.

TILE GRABBER

The tiles drawn on the screen were created as 32x32 pixel icons with Photoshop 3.0. They were saved as 24-bit images, converted to System Palette, and imported one-by-one into Director 4.0.4. After a while, this became quite tedious. It took several days to create just a dozen finished tiles, and I wanted to create many hundreds. Although this did not come to pass, I did succeed in creating a special tool that automatically cut, converted, and registered each tile from a special PICT file. This involved creating another custom Director XObject that could copy an image on the screen into the Mac clipboard. In essence, this XObject acted like a "screen capture" utility under Lingo control. What made this ability especially exciting was a new command in Director 4.0.4 called `clipboardtocast`. The trick was just getting the right data into the clipboard. Another Director command, `setregpoint`, made registering each icon easy. A nice side effect: The 24-bit PICTs were automatically converted to 8-bit System Palette by Director.

The Tile Grabber is not a stand-alone Director project, as was the Map Editor. The Tile Grabber is a special script that resides within the Map Editor that performs the following steps:

1. Loads a tile template image onto the screen, known to be at a particular screen location.
2. The XObject copies a 32x32 pixel block from that template into the Scrap (programmer jargon for the Clipboard)
3. The tile clipping script issues the ClipboardToCast and SetRegPoint commands.
4. The next 32x32 pixel block is chosen
5. Go to step 2 until finished.

The tile template is simply a 24-bit PICT file that has a number of 32x32 pixel tiles arranged on it. The tile clipper script assumes that the PICT conforms to this format. By using this template in conjunction with the tile clipper, I eliminated forever any registration problems I was having. Clipping and setting the registration point of hundreds of tiles is very hard on the eyes. This solution made is possible to do away with the manual operation altogether, and allowed me to concentrate more on drawing tiles in the few days left.

There was a mysterious bug in my Tile Clipper code that caused it to crash on non-PowerMacs. I have since traced this to a stupid programming error in which I told the computer to use a certain chunk of memory, but didn't tell it where it was. The result is random system crashing. The reason why it crashed only on non-PowerMacs is probably due to memory organization differences within the machine. If I had used the screen capture XObject for only tile clipping, this bug wouldn't have mattered as much. However, I used it in two places in the final game for a special transition. The computer would crash if it wasn't a PowerMac, which made it difficult to demo. Fortunately, I was able to borrow two PowerMac 7100s for the Thesis Show.

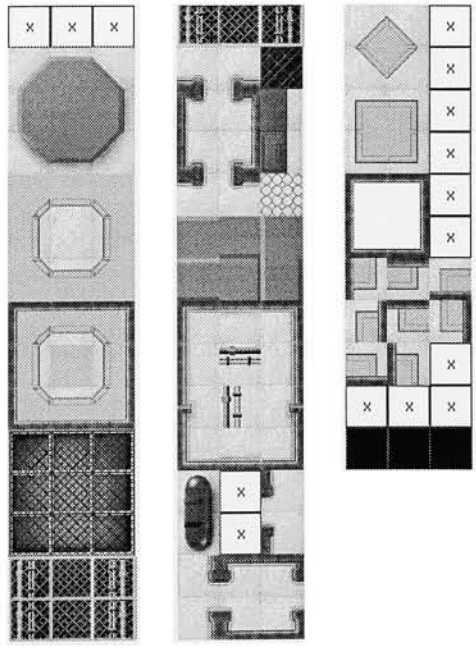


Figure 7: PICT Template

This 32-bit PICT file consists of 32x32 pixel tiles in a fixed grid. The TileGrabber code cuts these right from the PICT file and puts them in the Director cast.

SPACECRAFT MODELS
HSTV Space Plane

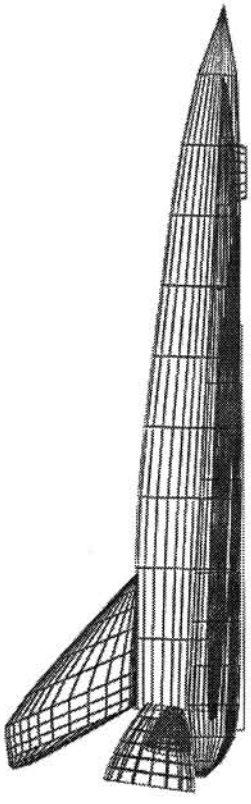
The HSTV (High Speed Trans-atmospheric Vehicle) is a second-generation shuttle concept. Like the Space Shuttle, it is a reusable craft used to put a payload into low earth orbit. Unlike the Space Shuttle, this vehicle is capable of taking off from a regular runway facility and boosting itself into orbit. It combines the function of the "Delta Clipper", a McDonnell-Douglas hypersonic aircraft that could theoretically fly from Tokyo to New York City in about two hours. Such a craft flies at fifteen times the speed of sound at extremely high altitudes. A rocket boost could conceivably propel a craft like this into low earth orbit.

The HSTV in the game is a commercial craft that must operate in atmosphere as well as in space. Thus, it has a sleeker, more aerodynamic look. It is a two-deck ship, split between payload and passengers.

This model mesh was created as a 3DS "fit shape" for the fuselage. The 3DS fit function allows you to specify a side and top 2D shape, which are used to generate a 3D object. The wings and tailfins were also created in this way.

The surface color maps were made with Photoshop 3.0. The fuselage is cylindrical mapped with a TIFF image file, while the wings are planar mapped separately. The planar mapping shoots through the entire shape, so the image appears on both upper and lower surfaces of the wing without any additional effort. Because of this effect, though, I had to avoid using any text or numbers on the color map, since they would have read backwards on the bottom side of the wing.

To make the texture look more aged, I applied specularity and bump mapping. The specularity map makes different areas look "shinier" than others, roughly corresponding to where the plates stand out. The bump map gives the impression of depressed plates and joints. I also airbrushed in some color variation into the color map. A glow map was used to make the windows stand out in deep shadow.



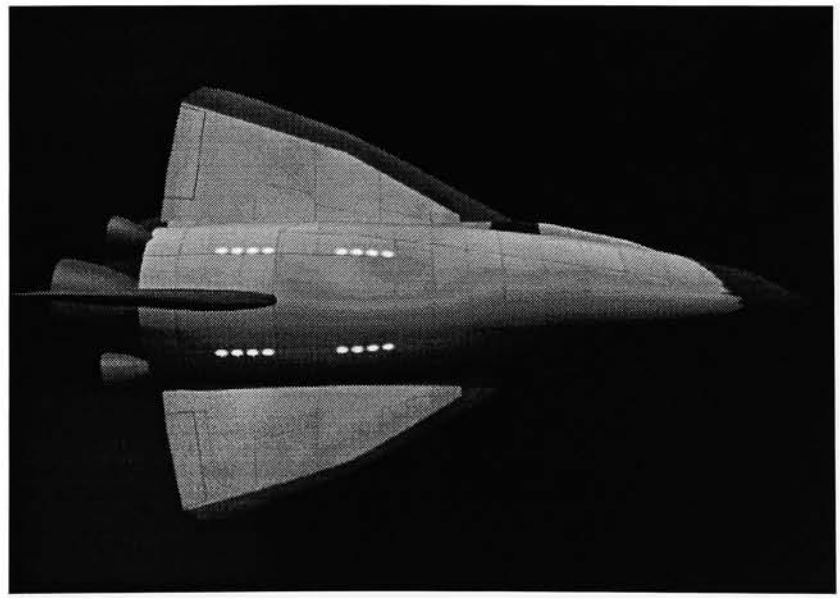


Figure 8a,b: Space Plane

Space Station

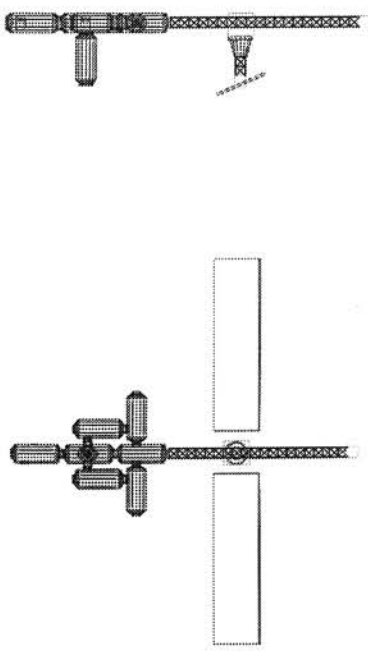
Based on NASA space station concepts, the "Charleston" (as it is named in the game) is a modular design that exists in low earth orbit. It follows the "Power Tower" design, in which a central post pointing down towards earth supports solar panels, away from the modules at either end. Such a design is also "gravity gradient stable", which means that the mass configuration of the design tends to stay oriented that way in orbit. The "gravity gradient" is the difference in the Earth's gravitational field strength. The farther one is from the Earth, the weaker the gravity. For a space station in orbit, the slight difference in gravitational attraction can make a difference in its long-term stability, unless it has more mass closer to the Earth.

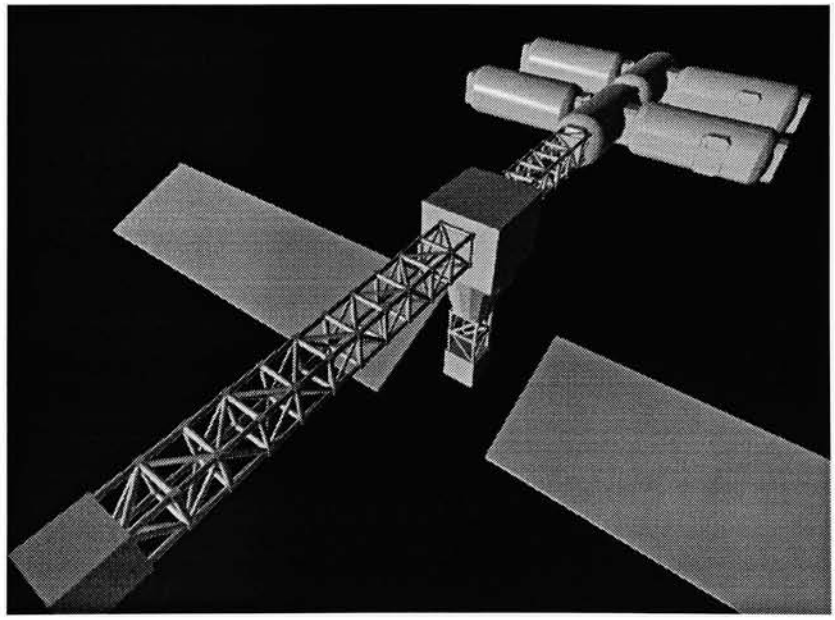
The station has a number of modular sections, which can contain crew quarters or scientific labs. Each section has multiple exits to other modules, which is something of a safety feature. The configuration of the bottom "docking module" was added later to provide extra clearance for the Space Plane, away from the rest of the modules.

The station was one of the first pieces modeled in *3D Studio*. I first created trusswork based on a NASA-designed structure. Each joint in the truss is joined to six others, which gives the structure great strength. After creating the basic truss unit, I created the cylindrical modules. These two components make up the majority of the station parts. The rest of the parts are simple geometric shapes that required no special modeling effort. The truss and cylindrical module are reused in many of the other modeled elements described below.

The texture mapping of the space station uses the same technique described in the Space Plane section. Cylindrical mapping was used on the modules. Bump maps, specularity maps, and glow maps were used in a similar fashion.

Figure 9: Space Station Geometry





Orbital Transfer Vehicle

The Orbital Transfer Vehicle is the backbone of Earth-orbit transportation. Although incapable of atmospheric flight, it is useful for cheaply moving a payload to a higher orbit (hence the name "orbital transfer").

The OTV design here centers around reusability of stock modules and trusses, with a few rockets strapped on for good measure. An OTV consists of a Space Station module attached to a round Control module. The round Control module, I reasoned, would be a space-manufactured component. It's self contained, with all the control circuitry and life-support necessary for a two-person crew. The spherical shape is efficient for pressurized structures (as in an ocean-going bathysphere) and may be more easily-constructed in the zero gravity environment of space. The truss provides the real stress-handling structure for the OTV, so the main rockets and thrusters are anchored directly to it.

Figure 10: Station & OTV

An image showing both the space station and the orbital transfer vehicle.

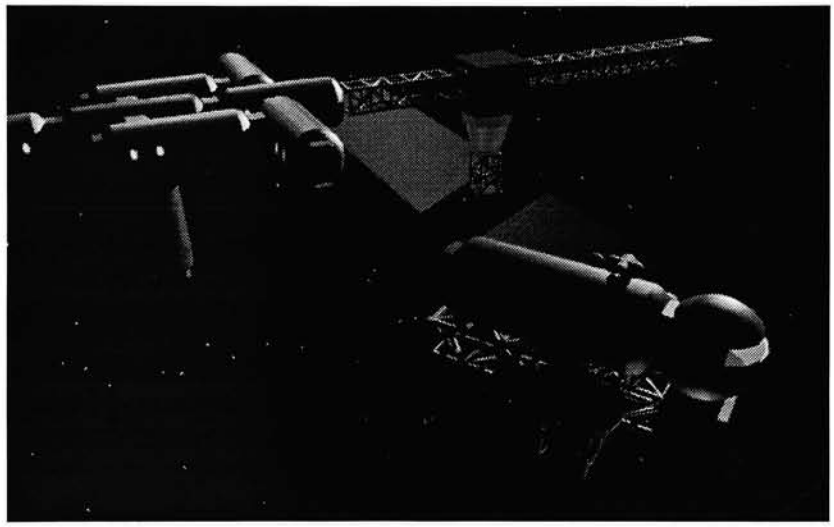
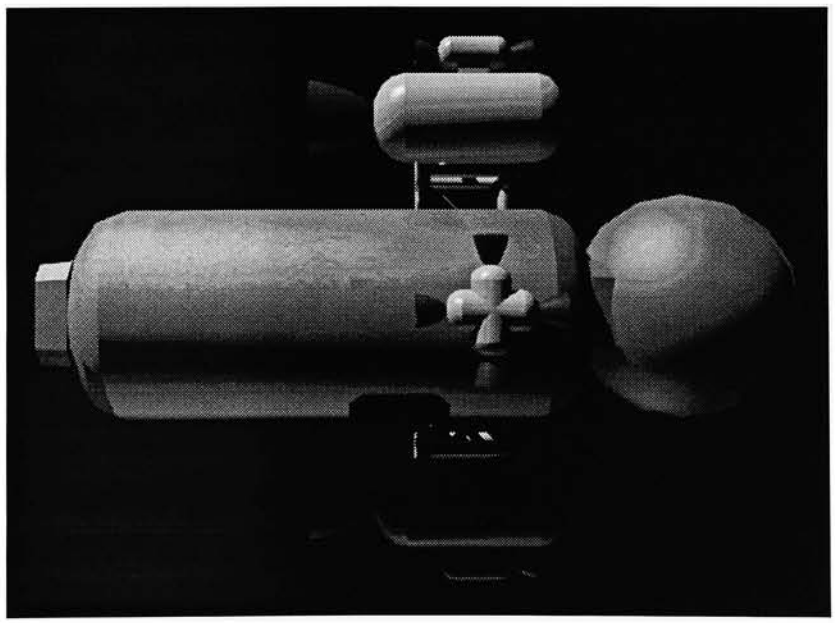
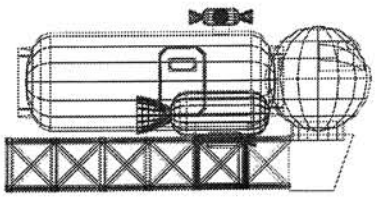


Figure 11: Orbital Transfer Vehicle





To create the Control module, I started with a polygonal sphere. 3DS is a surface modeler, and doesn't support constructive solid geometry per se. However, it does have useful "boolean operations", which allow one to subtract one shape from another. In this case, I subtracted a flat box from the front of the sphere to create the cockpit window. I created a translucent orange surface that had a slight ambient component. This makes the window glow as if there was light inside the OTV.

I created various thruster nozzles for use with the OTV. They are positioned roughly to provide maneuvering control for all six degrees of freedom. The center of mass for the OTV was assumed to be somewhat forward of the geometric center. The thruster nozzles were created by deforming a regular cone, hollowed out using boolean operators.

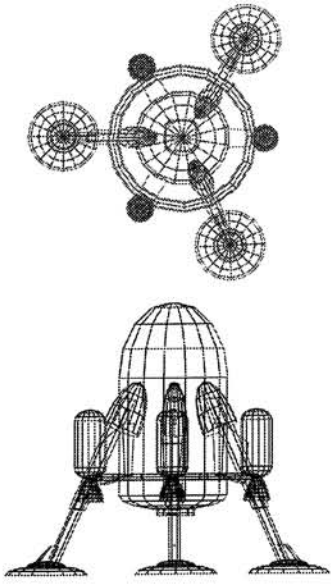
Lunar Lander

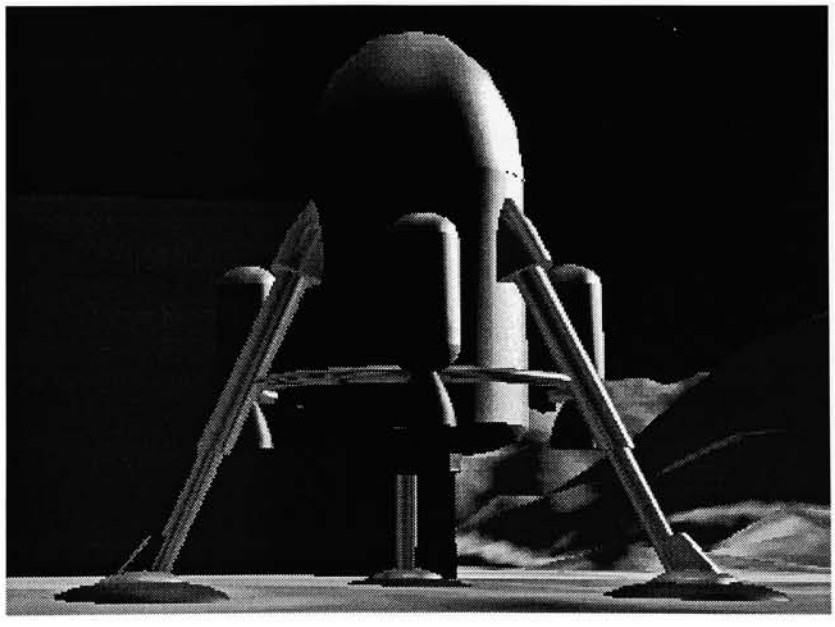
The moon lander is a single-purpose vehicle. It can rendezvous with an OTV or station in orbit around the moon, take on passengers or cargo, then land on the moon. It is also used for short hops around the lunar surface, when a Lunar Rover (see below) can not be used.

The design for this particular ship is based on a standard station module, except it has a cone on front end. The module is also cut-down in size, which gives it a squat appearance. The landing gear design is based on an early NASA concept, pointed out to my by a thesis advisor. The Lander is designed to accommodate some more of those standard rocket modules used on the OTV. They are, however slightly smaller.

I chose to apply a coppery texture to some components of the lunar lander. It gave the craft something of a retro accent, though I'm not sure why it had that effect on me.

Figure 12: Lunar Lander

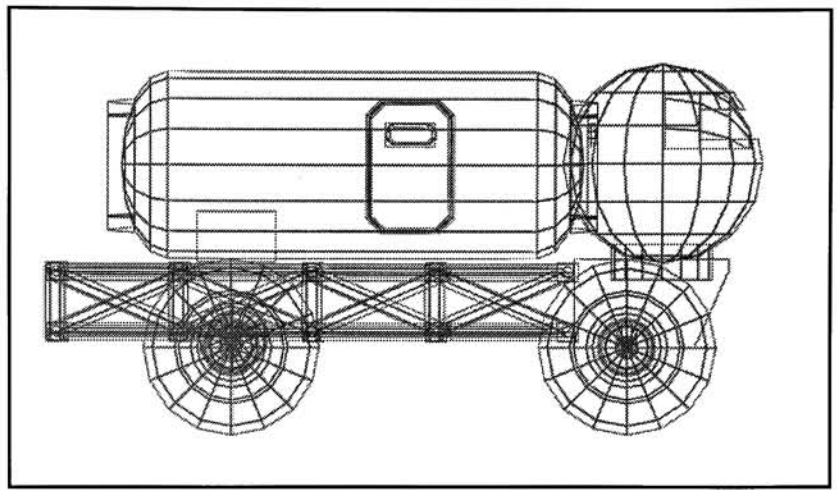




Lunar Rover

This is a large, truck-like vehicle based on the OTV module parts. The truss, passenger module and control module are lifted directly from the OTV design, with big tires in place of rocket pods. The control module is mounted forward of the front steering wheels, primarily to match the motion in an animation that was created earlier. The way the "through the windshield" view is animated makes sense only if the steering wheels were behind the driver (more on that later).

Figure 13: Lunar Rover



MOONBASE

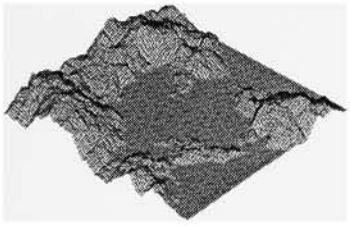
The moon base is designed around a center administrative core that was soft-landed on the moon in pieces. Construction of the surrounding "rings" of buildings started with the laying of a concrete foundation, which are visible as circular rings. The moon base is further divided up like a pie, with commercial, research, and military slices making up the whole. Several observation tower/beacons are placed strategically around the base and its outlying posts.

The moonscape is a model built with the help of *VistaPro 3.0*. This is a landscape generator, capable of created all kinds of mountainous terrain, complete with simulated trees, clouds, and oceans. To create the moonscape, I generated a landscape that could be "flooded" with water. I wanted a ring of mountains to suggest a crater wall, as the moon base is supposed to be located in the Tycho lunar crater. After some trial and error, I generated an appropriate mountainous surface. I then applied VistaPro's "lake" function, which created a crater-like geometry. This I exported as a 3D DXF file, suitable for use with *3D Studio.*

The geometry did not prove sufficient. The utility I had (*GIFto3DS*) limited the size of the landscape to 256 units by 256 units, which was far too coarse to be effective even with smoothing. VistaPro does a good job of making realistically-lit mountainscapes, so I decided to use this as the basis of a texturemap. I had to take an aerial snapshot of the *VistaPro* landscape, simulating a 500mm lens to flatten out the perspective, and map this on top of the geometry in 3DS. I assigned all the colors in the world to shades of gray to make it "moonlike", and then enhanced the resulting image with Photoshop 3.0.

Figure 13a, b: Landscape

The major features of the landscape can be seen on both the VistaPro (right) and derived 3DS geometry (below).



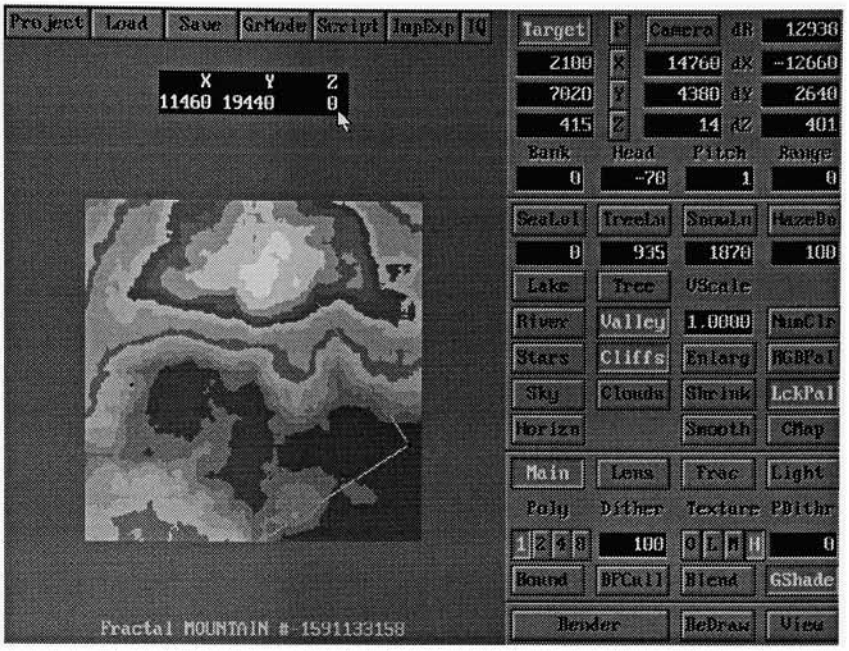
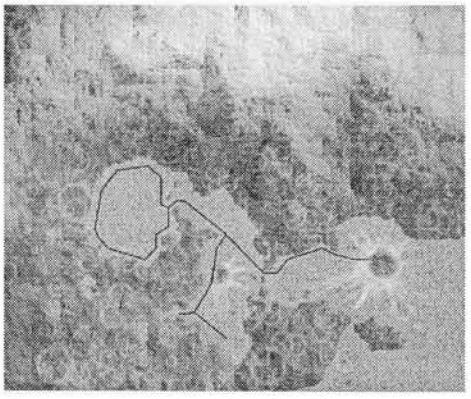


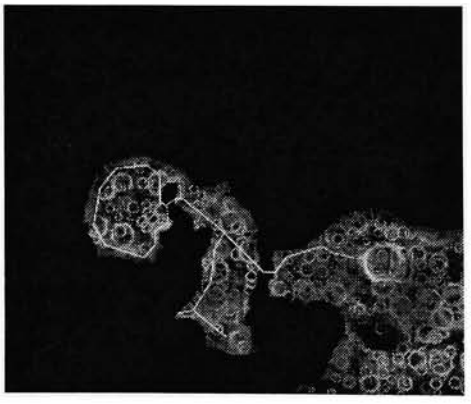
Figure 14a-c: Texture Mapping

The color map and bump map are mapped over the geometry and rendered in 3DS (next page).

The snapshot was used to make a 3DS texture. To create the illusion of many smaller craters, I used 3DS bump-mapping. Crater impacts on the moon tend to overlap each other, so it was simple to make a black and white "crater map".



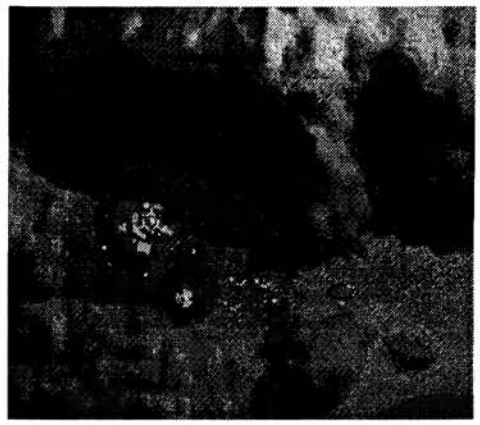
To complete the illusion, I painted some additional color into the texturemap. The moon is mostly gray, but there are slight tinges of light tan moon dust. Craters also have "ray systems", which look like white streaks emanating from the center of a crater. I also added a road to serve as the "track" on which the lunar buggy would run.



For effect, I also added a very slight layer of fog in 3DS. While there is no atmosphere on the moon, the fog layer adds a dusty appearance to the crater floor.

The moonbase buildings were constructed on top of the simulated moon surface within 3DS. The ring-like design was actually inspired by a computer engineering book about microprocessors. Modern microprocessors have sev-

eral "levels" of security, often described as "rings of protection." This image in mind, I created a base floorplan that resembled rings. The hub of the base is "ring 0", where administrative tasks take place. The outer rings are used for housing, scientific research, and military activities. Scattered further out from the rings are the landing pads. Some of the military facilities have their own pads. The main landing pads, though, are located some distance away from the main base structure. There is also one outlying post between the main landing pad and base, which I imagined to be some private research facility.



The buildings are relatively undetailed, using variations on the basic 3DS textures. As the player never gets a chance to get really close to buildings, this lack of detail is not too noticeable. I relied primarily on lighting to get a sense of scale, but there is still a disturbing toy-like quality to the output quality. I attribute this primarily to a mismatch in texture resolution. The moon's surface, since it is so large, has to stretch out its texture map over a wide area. This gives the surface a grainy, out-of-focus texture when viewed at near distances. The moonbase buildings, on the other hand, cover a much smaller area. The textures remain sharp at close range, while the moon surface goes fuzzy. The effect is that the buildings have a machined, toy-like quality. They stand out from the fuzzy background of the surface, and look more artificial. There are two ways to correct this. I could have retextured the entire moonbase building set (preferable), or apply a higher-resolution texture map to the moon's surface. I chose the latter. Unfortunately, the maximum resolution I could pull out of VistaPro was not great enough to eliminate the stretching effect.

ANIMATION DESIGN

The 3D spacecraft and landscape models were used to create animations within 3DS's *keyframer* module. This part of the 3DS program provides excellent control, with fast tools for fine tuning the final animation.

I was able to apply aesthetic control over the particular kinds of motion I wanted to portray in four animation sequences. These were "ToStation.Moov", "ToMoon.Moov", "Lander.Moov", and "ToBase.Moov". These sequences were originally in the Autodesk FLC ("flick") animation format, which is the native output format used by 3DS. They were converted to QuickTime using Equilibrium Technologies' *DeBabelizer* .

Space motion dynamics

Motion in space has quite a different "feel" than atmospheric flight. The game's level of technology uses limited-fuel chemical rockets, hence courses are plotted for maximum fuel efficiency. Uses of thrust tend to occur in controlled directional bursts, which results in somewhat jerky feel to the motion as the spacecraft floats, applies directional thrusters, then fires rockets. The sense of speed in space is also diminished. Although a spacecraft operating in low earth orbit has a speed of 7 miles/sec, there is no immediate reference point. Relative speeds between orbiting spacecraft are low, on the order of feet/second. The physical laws of motion are also quite evident with the lack of friction and gravity: objects in motion stay in motion unless an external force is applied.

I tended to use these observations as guidelines in designing the animation. The most accurate portrayals are shown in "ToMoon.Moov" and "Lander.Moov." In the former, an OTV is shown launching from the station, destined for the moon. Directional thrusters fire as the OTV maneuvers away from the station. After a few seconds, the main rockets fire, and the OTV accelerates to the Moon. In the latter movie, a lunar lander is shown descending toward the camera. Just before it impacts, three rockets fire to slow its descent. The weak lunar gravity slowly pulls the craft to the ground.

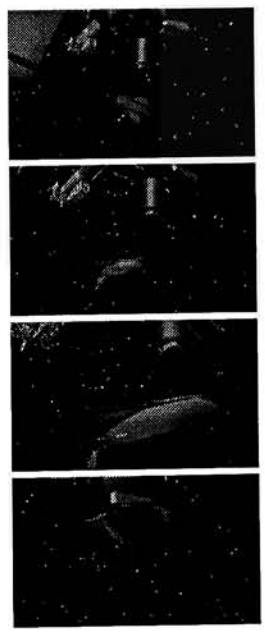


Figure 15: Space Plane Docking

Stills from the animation (original size: 200 x 120 pixels).

I took more liberties with the Space Plane docking movie, "ToStation.Moov." The Space Plane flies more like an aircraft, banking into its turns with fuel-wasting continuous thrust adjustments. The rationale behind this was twofold: first, I reasoned that a trans-atmospheric craft might be better to fly if its controls reacted similarly in both atmosphere and space. More importantly, it just looked better.

For the final "ToBase.Moov" animation, I depicted a first-person "drive over the lunar surface" as the players are driven to the lunar base. The view dips up and down as the lunar buggy bumps over the cratered lunar landscape. At times, the buggy appears to get stuck in ruts as it negotiates the rocky terrain. This was a rather challenging animation, as I had to animate both the camera's position and direction. I had to keep the camera at a consistent height over the uneven surface, avoid plowing through rocks, and simulate the bumpy ride in a

realistic manner. This was done by trial and error. I plotted a rough path through the lunar landscape that avoided the major mountains, and drew this as a road on the lunar texture map for guidance. For the smaller hills and ruts that invariably interrupted the path, I maneuvered the camera so it either went over or around the obstacle. The road I plotted meanders past interesting landmarks. The base is visible in the distance for much of the journey, building some sense of anticipation. The moon rover also drives through some shadowy areas, which looks neat.

Camera Positioning

For all animations, I tried to be careful about camera placement and tracking. It was important that I maintain smooth motion. The camera motion uses ease-in and ease-out to avoid the discontinuous "sudden movement" flaw seen in some computer-generated animation. I tried to avoid camera motion for its own sake, limiting myself to slow "tracked" shots for subtle shifts in angle. This is most evident in "ToMoon.Moov." The camera's position is shifted slowly downwards as the OTV passes by, which imparts a kind of ponderous rolling sense to the shot. The OTV is taking its time. The camera ends up slightly below the OTV just before it kicks in the main thrusters and accelerates towards the moon.

Cameras were also placed to show either "arriving" or "departing" actions. Both "ToMoon.Moov" and "Lander.Moov" are examples of this.

For the moonbase surface shots, I had to choose camera positions that would not show the edges of the model. The simulated moon surface is quite small, so I had to position the camera so that mountains loomed in the background for many of the animations. The world ends abruptly if this care isn't taken.

Editing

There is only one scene with two shots: "ToStation.Moov." The scene opens showing something arriving. As it comes closer, we see it is a space plane. It flies directly in front of the camera, giving us a good look and satisfying our initial curiosity. As it flies away, though, we see the space station itself in view as the plane banks towards it. A cut occurs, showing the plane heading towards the underbelly of the station. The shot transition takes place so the space plane is roughly in the same third of the screen in the beginning of the scene. An earlier version did not do this, and there was a noticeable "jump" in the flow of the scene.

Lighting

In each animation, a single light provided strong directional lighting, placed to accentuate the shadows of the spacecraft and station. For some shots, this single source did not prove adequate for illuminating moving spacecraft. For these situations, I followed the advice of an old friend and "lit it so it looked good." The Space Plane has a weak omnidirectional light locked directly overhead it to make it visible. The space station's solar panels have lights dedicated to making them shine. The OTV and lunar lander have lights placed to accentuate their edges with highlights. Overall, I favored a dark, shadowy look, so most lighting is devoted to providing edge lighting, with a touch of fill lighting to bring out the shape.

Effects

The starfield is generated by a built-in 3DS IPAS plug-in, and required no special effort on my part. This starfield, unfortunately, isn't as dense as it should be in space. The 3DS plug-in simulates the view from the earth's surface, so the star visibility is reduced by Earth atmosphere.

The rocket effects were created by using an opacity-mapped cone, as described in the book *Inside 3D Studio R3*. The top of the cone is 100% opaque, but it becomes increasingly transparent towards the bottom. Orange omnidirectional lighting was placed near the cones to cast bursts of light. These lights were range-limited to prevent them from illuminating the entire scene (particularly important in "Lander.Moov" and "ToMoon.Moov.") 3DS allows you to hide and unhide any object during the animation, so the cones/lights used are actually part of the model.

Conversion

3D Studio saves its animation in the FLC format, which is not native to the Macintosh. *DeBabelizer*, however, can convert this format to QuickTime.

Each scene was rendered separately. One movie, "ToStation.Moov", consists of two scenes. *DeBabelizer* was used to clumsily edit these together. Originally, I had planned to use Adobe *Premiere*, but I had an ancient version of the program. This version was unstable on the PowerMac 7100/80AV I was using for my development machine, so I had to use *DeBabelizer.*

Framerate and Size

I chose an 8 frame-per-second (fps) rate for all animations. Since the motion of objects within scenes is fairly slow, the low frame rate works fine. With an 8 fps framerate, I could render fewer frames of animation for a given length of time. Additiionally, 8 fps is quite achievable even on low-end Macintoshes. I stored each final animation using 24-bit Cinepak compression, which is an efficient QuickTime codec for animation purposes. QuickTime movies produced with this compression scheme are both small and fast. I limited the size of the animation window to 200x120 pixels, which conforms to one of the standard "wide screen" movie aspect ratios.

Sound Effects

The problems with Premiere 2.0 prevented me from inserting a soundtrack directly into the movies, so I used Director's built-in sound capabilities. For synchronization, I checked **the movietime** digital video property for each animation as it played. When a pre-defined time is reached, an appropriate sound was triggered by a simple Lingo script.

Sound effects were both synthesized and sampled from CD. Most of the "click" sound effects are from my collection of self-made sounds, with additional tweaking in Macromedia *SoundEdit 16* to make them less familiar.

Of the sampled sounds, the "whooshing jet" noise in "ToStation.Moov" is a pitch-shifted version of a 747 takeoff, roughly synched to the animation. The "thruster burst" sounds are synthesized white noise, shaped with the

SoundEdit "envelope" function. The "OTV docking release" noise is the sound of a can opener, pitch-shifted and played backwards and ended with a THUNK noise taken from the game *Doom*. The SoundEdit 16 "bender" tool is used to make this effect pitch higher as it plays to the end. The "collide with object" effect is the sound of metal CO2 cartridges, lightly tapped together to produce the ringing "TINK" noise. Some of the sound effects might seem out of place when described, but in testing people tended to just accept the sounds as they were accompanied by new visuals.

There is no transmitted sound in space. However, structural vibrations can transmit sound into your space suit or spaceship, which could be heard. What about the whooshing jet sounds? It would be kind of disappointing if that wasn't there. Compare "ToStation.Moov" with "ToBase.Moov" to experience the difference.

For the ambient background noise in each location, I created a looped sample that tried to capture that "high pitched whine" noise you experience in commercial jets. In retrospect, this might not have been a great idea, as it just proved to be annoying. In any case, I reasoned that the noise was a combination of electrical hum and structurally-transmitted turbine noise. I had also read once that military power systems run at frequencies higher than 60Hz. I chose 400Hz as a base tone in SoundEdit 16's "FM synthesizer." I mixed in other harmonics of this base frequency at 200 and 100Hz to "fatten" the sound. On the lowest frequency, I applied a 1Hz modulation frequency, which makes the sound throb once each second. Each frequency occupied its own track in SoundEdit 16, and was mixed around to match the characteristics of each location. The Space Plane is relatively muffled. The Space Station has more reverb, as it is a larger space. The OTV has the most annoying sound, as it is stuffed with electronics and computer systems.

Director Effects

This section discusses some of the Director-related aspects of this project. Most of the information here is related to scripting, though the organizational facets of the design are mentioned as well.

2D Map Editor Reuse

The 2D Tile display in the game uses, as was hoped, large portions of unmodified code from the Map Editor. Many of the interface features of that program, such as the "point-and-click" tile selection interface, were very useful in determining which tile the user was clicking on. This was used only once in the final game. A red button in a vehicle could be used to active the launch sequence, under the right circumstances. Other code, such as the nifty scrolling code, was practically dropped-in. The only modification was resizing the tile display to use 11x11 tiles, rather than the 15x15 setup used in the Map Editor.

Console Window Text Field

The Console window is used to enter commands. You can also talk to other people in the game using this window. The Console also remembers everything that was displayed in it. Since the Console window can show only 5 lines of text, a scrolling function was added. To hide the ugly scrollbar, a special *rollover* script was written to reveal it only when the mouse cursor passed over that section of the screen. Double-clicking the text field erases all past text.

Animation Text Overlay

The Animation window shows QuickTime or PICT images with a text field overlaid on top. The text field has a transparent background, so animations can play and still be seen. While the animation is playing, the contents of the text field can be manipulated with various text effects. I used several effects, based on some rather old Applesoft BASIC text routines I wrote years ago. Director provided some nice features, such as LEFT/RIGHT/CENTER text alignment. The "teletype" effect was done by stepping through the characters contained within a text string, updating the field with more and more text over a period of time.

Console Command Scripting

Special commands, such as JUMP, can be typed into the console. In addition, players can communicate with each other by just typing a sentence into the window. A special script attached to the console window waits for the RETURN key to be hit, which signifies that the user has entered text. If the first character of the text is a"/" (slash) character, then the sentence is assumed to be a command. If there is no slash, then the sentence is broadcast to the other player's computer as something that was spoken aloud. This text appears on both computers.

The Lingo chunk expression commands were used to further parse a command sentence. After removing the first "/" character, WORD 1 was considered the command, with WORD 2, WORD 3, and so on considered the parameters of the command. For example, consider the command JUMP. There are no parameters in this case. A simple IF-THEN comparison is made with WORD 1. If it matched JUMP, then the script performs some special action under the hood. For the sentence /NAME David, the command is NAME. The parameter (WORD 2) is David. The script does something to update the name of the current player.

COMMAND LIST:

/NAME <NewName>	Choose a name for yourself. The other player is also informed of the new name, and the computer precedes every sentence you "say" with this name.
/ME <Action>	Do something. The actions are not understood by the game, but you can use this command for dramatic effect. The computer just prepends yours name to the <Action> line you type. If your name

	is DAVE (as set by the /NAME command), and you type "/ME jumps and sings" into the console, the computer responds with "DAVE jumps and sings." It's up to the player to construct a meaningful sentence.
/WHERE	Tells you where you are in the game world.
/HELP	Gives you a brief list of commands.
/JUMP	Computer responds with a flippant remark
/SCREAM	Computer responds with a flippant remark

Character Tracking

A special parent script is used to keep track of the positions of the characters within the game. Since there are two players, on different computers, this script must keep track of:

Name of Player:	The name, as assigned by the /NAME command in the console window.
Current Map:	Which "map" that player is currently exploring
Current X,Y Location:	Where on the map the player is currently standing
Alive:	Whether the player of the OTHER computer was present or not, and needed to be tracked.
Health, Status, Inventory:	Other attributes of the player. These were not implemented in the thesis game, though the code is partially there.

This parent script is used to make instances for both players, on each computer. The characteristics of each player are updated via the serial port connection.

Serial Communication

Each PowerMac is synchronized with each other via "null-modem" serial cable. This allows each Macintosh to communicate directly with each other at a fairly high data rate (38400 bits per second, in this case).

XObject

The Director built-in SerialIO XObject is used to control the Mac's serial ports. Because this is a high-speed (greater than 9600bps) connection, hardware handshaking (a special connection signaling technique) was used to ensure data transmission. Without hardware handshaking, bits can get "lost", which leads to errors and unreliable operation. The use of hardware handshaking has an unfortunately side effect. Once the serialIO XObject is activated, it has to be left active. If shut off, the serial port immediately "drops DTR signal" and causes the other Mac to freeze, waiting for that signal to be reasserted. It can only be broken by either restarting the serialIO, or by physically breaking the connection between computers.

The SerialIO XObject accepts entire strings to be sent, as regular ASCII, over the serial port. Although this XObject has some error checking, it is not smart enough to ensure that the entire string is received as one. It can be received as a partial fragment, depending on what the serial port on the receiving end is doing. I had to write special code to force this to happen as whole strings.

Serial Initialization

When the game starts running on a computer, it initializes the serial port as described above, and immediately sends out message that says "HELLO! My name is NoName, and I'm on MAP 1 at location 10,10." It then waits for a response from the other computer. If there is none, then the game just assumes that nobody else is there and you play by yourself. If the other Mac is already running, it responds with a *"HELLO NoName! My Name is XXX, and I'm on MAP x at Location X,Y"*. In this way, each computer is brought up to date.

Game Synchronization

Generally, the communication between the two computers is in the form of special commands. The first word is the command, and subsequent words are the parameters. In the Startup example above, the HI, NAME, and POS commands were used. Note that the player doesn't issue these commands directly. These commands are used signals between each computer so they keep in step.

Example of the HI command string: HI Fred Map1 10 10

There are rules established for each command. If a computer receives a HI command from the other, it is required to send a NAME followed by a POS command in response. This is called the acknowledgment.

There are other commands that perform special functions. The OTVGO command tells the other computer that someone has pressed the LAUNCH button. The BYE command tells the other computer that a player has quit or left the game. This is important because a missing player shouldn't be drawn or updated on the screen. CLICK tells the other computer that a particular tile was clicked. SYS is a general status message that can be used for any purpose.

The scripting is very straightforward. The only tricky part is keeping track of what all these arbitrary commands do, and have the computers react in an appropriate way. Every time a player moves on one computer, a POS (position) command is sent to the other. This way, both computers know the position of the players in the game.

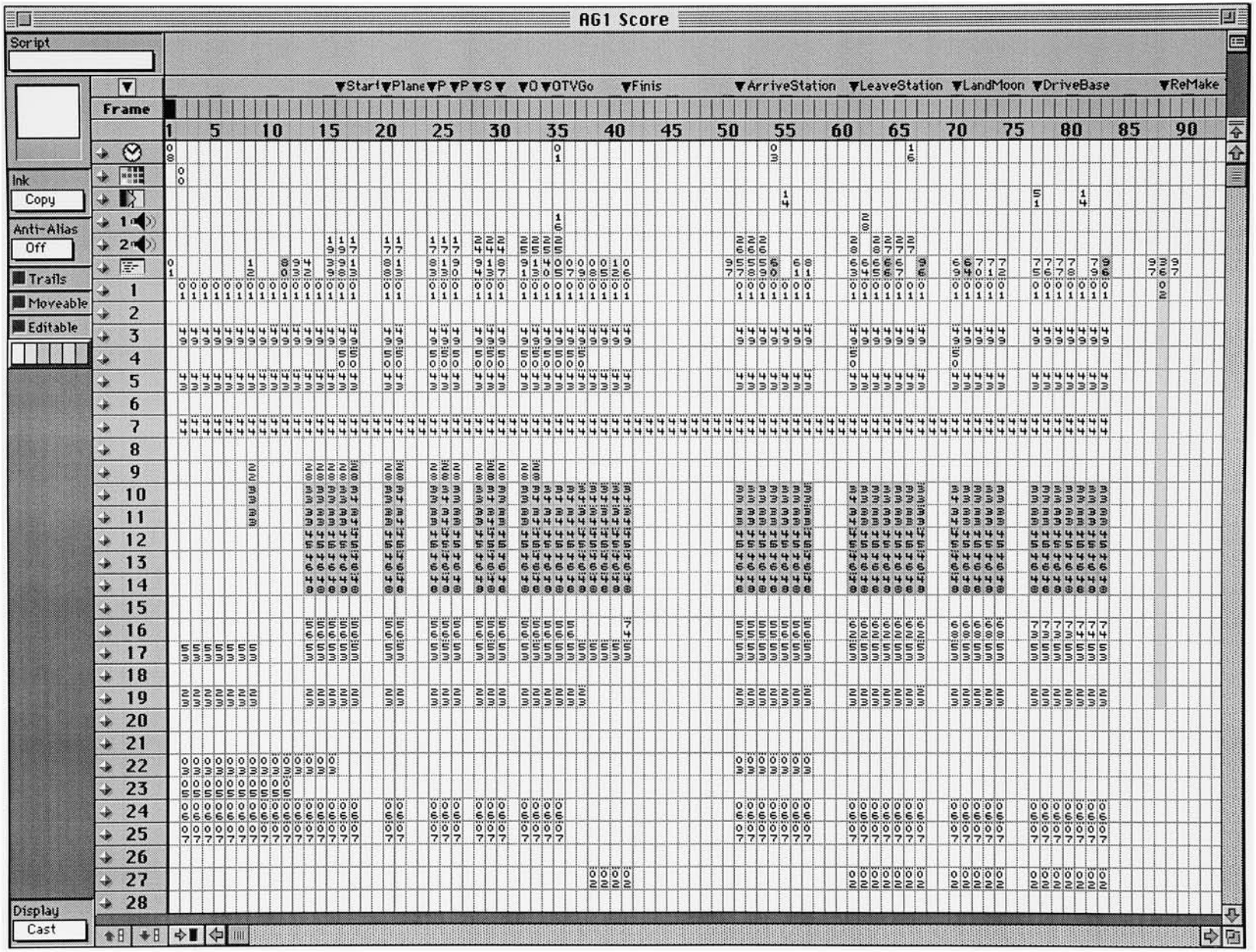
Movie Organization

The game is structured as a number of linked interactive segments within a single Director 4.0 movie. Several frames are devoted to initialization of variables and graphics prior to an event loop. The event loop checks for keyboard and mouse activity, and dispatches the appropriate script as needed. There are several sections, one for each interactive map (space station, plane, etc.) There are also several frames devoted to playback and synchronization of animation before and after the interactive portions of the movie.

All tile, character statistics, and serial port functions are embodied within instances of various Parent Scripts. This object-oriented approach allowed me to encapsulate related functions within a single code entity, which makes for cleaner program design. The alternative would be to create dozens of handlers at the Movie Script level, which is becomes difficult to manage. Parent scripts also force data encapsulation, in which variables that are associated only with a certain kind of object are ALWAYS linked to that object. This helped to eliminate the creation of dozens of global variables.

OVERLEAF
Figure 15: Director Score Window

Organization of Director Project.


AG1 Score
Script
Ink
Copy
Anti-Alias
Off
Trails
Moveable
Editable
Display
Cast
Frame
▼Start
▼Plane
▼P
▼P
▼S
▼O
▼OTVGo
▼Finis
▼ArriveStation
▼LeaveStation
▼LandMoon
▼DriveBase
▼ReMake

User Feedback

The culmination of the thesis project took place during the Thesis show, which was well attended. Several groups of people tried out the game and gave me feedback.

Command Line Interface

The Console window, which is used to communicate with other players as well as enter commands, took some explaining. Since the mouse could be used to click on various portions of the screen, most people assumed that this was a mouse-only game. I had to direct their attention to the keyboard and Console window, as this is where the richness of the game environment starts to manifest. Contemporary applications make little use of keyboard entry (especially on the Macintosh), so this style of interface was not "intuitive" for most people.

Movement Control

The player's character is controlled by a "movement diamond" in the lower middle of the screen. You can move "north" by clicking on the top part of the movement diamond. The screen responds by scrolling the entire background, leaving your character centered in the middle of the screen. This did not seem to confuse anyone who tried the game, though many expressed the desire to control movement through the keyboard. Many people, accustomed to a Mac-style interface, tried to move the character by clicking on the character's icon on the screen. This would have been nice to implement a system that allowed exclusive use of keyboard and/or mouse. The mouse seems to encourage a more exploratory approach, while the keyboard is all business.

Character Design

For the purposes of the thesis show, I created four small 32x32 pixel sprites to represent the characters. There were actually only two distinct sprites. One was of a man in a jumpsuit, and the other was of a person in a spacesuit. Two different colors (red and green) were used to distinguish different players. Players of the game complained that the characters were too similar, and that more varied shapes would be more appealing. The thesis show version of the game, unfortunately, did not have the ability to designate a character shape. This would have been a nice touch, if additional program logic was built in to handle the problems of two players choosing the same graphic.

Interactivity

The thesis show version of the game had fairly limited interactivity, at least in the classic "point-and-click" sense of the word. Although the current game engine is capable of detected where and what tiles are clicked on, this is not used to great effect. When a player clicks on the icon of the *other* player, that other player sees "<player's name> looks at you" as kind of a silly action. The other example of click detection is the OTV launch button. Either player can launch the OTV to the moon, but only if both players are on board.

As far as communication between players go, the game provides a different kind of interactivity. Using plain text as a medium of communication has a very wide range of action possibilities, if only from the "let's pretend I did this" perspective. A more sophisticated graphics engine, capable of gesturing or allowing more emotive control of the player's alter ego, would be the next logical step.

ASSESSMENT

I met the minimum requirements to show off my game: interface design and functional design. With this, I was able to demonstrate the basic concept behind my multiplayer game. I also met some personal knowledge goals. I learned the basics of Macintosh programming and picked up 3D Studio at the same time. Both of these skills are of particular value in the entertainment software industry.

My initial idea was to create a storytelling environment, built on the social interactions of players against a 1920s / Space Frontier Adventure backdrop. The resulting "feel" of the game is far from my original concept. The moonbase and space station design are rather plain; not reminiscent of the 1920s at all. As pointed out by an advisor, this is not necessarily a bad thing.

Two technical goals were not met. The first, construction of a functioning computer role playing statistic system, was approached in only the most preliminary of ways. The second, implementation of a working Director XObject to allow AppleTalk network multi-play, was shelved due to time constraints. I did not determine whether it was in fact possible, though some initial coding was completed to test the concept outside of Director.

Engineering Large Projects

A great deal of effort went into non-visual activity, which was not my original intention. I spent a lot of time reading about space and space travel. I read a lot about the Macintosh as a technical platform, and spent quite a few hours just thinking how I might start to create the Ultimate Structure for my project. Perhaps I spent too much time worrying about the unstated goals listed in the beginning of this report.

The technical specifications I set for myself were subconscious. I did not think of another approach to the design, and fell into a standard engineering approach to software development. I feel that I should have spent more time "just creating" the stories and environments in which I ultimately wanted my game to take place.

Hacking Smaller Projects

Despite these setbacks, I was able to craft animations that I found visually pleasing. Although the same personal foibles tended to guide my creative work, I am happy with the way the animations came out. It seems that a more limited-scope task suits itself to my creative approach. The combination of my engineering-skewed outlook with my aesthetic senses appears to work well for small tasks, fueling my creativity rather than stunting it. When considering something of much larger scope, however, the engineering approach starts to work against me.

By concentrating on a purely visual project, I could have explored my creative side further. There was just one thing in the way: the "gee-whiz" aspect of my creative efforts. I am something of a perfectionist in my conceptual thinking, and part of that perfection is the "difficulty factor" involved. I used to measure difficulty in purely technical terms before...it is time that I start to really consider the visual/creative side more. Because there is no "right answer" in visual design, I think I find this much harder.

Although this project didn't meet the goals I set for myself, it contains elements of which I am proud. I successfully built a simple multiplayer game, based on Director 4.0 with my own custom-programmed extensions. I learned to use Autodesk's *3D Studio* modeling and animation package. I created a graphical user interface that I liked, without resorting to using someone else's textures or scanned elements. I also got to explore my own creative process, which will help me as I tackle future projects of this magnitude. The thesis project was a satisfying culmination to my graduate studies.

Reflections

The most surprising thing to me is that I don't feel bad about not meeting every aspect of my original design specification. While I do believe I could have achieved more, I am happy with those things I did accomplish. My greatest regret is not having actually made something that fit the spirit of the 1920s. I ended up with something that felt distinctly different from the description made early in this report. In essence, I have completed the first half of a greater design project: Tool Building. The next phase, as yet incomplete, is World Building. This was my original inspiration, but I met it only partially.

The building of a *multiplayer* environment was intended to create a *collaborative* storytelling process. It is, perhaps, technically and conceptually interesting, but it lacks the excitement and drive of a project fueled by a group of like-minded people. I believe the next stage of development would be group collaboration in the world building aspect. My original vision, to create a 1920s-style *finished environment,* is something like that of a theme park or bar. I provide the "built-in" ambiance, and the customers enjoy playing in it. Another approach would be to provide the tools and the space, and let the customers do what they want with it. The distinction between *playing in* versus *playing with* is a good one to keep in mind as expectations of interactivity continue to rise. It's no longer cool to be merely exploratory. This direction is one that I will explore further.

Dave Seah
August 20, 1995

BIBLIOGRAPHY

Books

science:

David Halliday and Robert Resnick. *Fundamentals of Physics, 2nd Edition.* John Wiley & Sons, 1981.

William K. Hartmann, Ron Miller, Pamela Lee, *Out of the Cradle: exploring the frontiers beyond earth*

Eugene Mallove and Gregory Matloff, *The Starflight Handbook: a pioneers guide to interstellar travel.* John Wiley & Sons, 1989.

Gordon R. Woodcock, *Space Stations and Platforms.* Orbit Book Company, Malabar, Florida, 1986.

art sources:

Carol Titelman, editor, *The Art of Star Wars.* Ballantine Books, New York 1979.

Deboral Call, editor, *The Art of Empire Strikes Back.* Ballantine Books, New York 1980.

LucasFilm Ltd, *The Art of The Return of the Jedi.* Ballantine Books, New York 1983.

design:

Patricia Bayer, *Art Deco Architecture: design, decoration, and detail rfom the twenties and thirties.* Harry N. Abrams, Inc, Publishers, New York 1992.

Brenda Laurel, *Computers as Theatre.* Addison-Wesley, 1993.

Raymond Loewy, *Industrial Design*

Jacque Legrand, *Chronicle of Aviation,* JL Internation Publishing Inc, Liberty, MO, 1992.

magazines / conference proceedings:

SIGGRAPH '94: *Digital Illusion: Networked Interactive Entertainment*

Brian Sutton-Smith. *Games.* Article supplied by Bob Keough

Programming

Apple Computer, Inc. *d e v e l o p CD-ROM*

Apple Computer, Inc. *Inside Macintosh 2nd Edition CD-ROM*

Macromedia. *X-Object 1.1 Development Kit.*

Apple Computer, Inc. *Inside Macintosh 2nd Edition.* Addison-Wesley Publishing, 1992.

Scott Knaster and Keith Rollins, *Macintosh Programming Secrets.* Addison-Wesley Publishing, 1992.

James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes, *Computer Graphics: principles and practice, 2nd Edition.* Addison-Wesley, 1990.

Comic Books / Graphic Novels

Yukinobu Hoshino, *2001 Nights.*

Masamune Shirow, *Appleseed.*

Science Fiction/Fantasy

Scott Cowley, *Spacecraft 2001.*

Isaac Asimov, *Foundation Trilogy.*

Harry Harrison, *The Stainless Steel Rat.*

Orson Scott Card, *Ender's Game*

Frank Herbert, *Dune.*

Animation
Wings of Honneamise
Porco Rosso - Legend of the Crimsom Pig

History/Historical Fiction
Saint-Exuberry. *Wind, Sand, Stars*

Films
Star Wars

Computer Games / Space Themes & RPGs
Muse Software: Moons of Titan
Interplay: The Bard's Tale
LucasArts: X-Wing, TIE Fighter
Microsoft: Space Simulator
Origin Systems: Ultima I-IV, Privateer, Space Rogue, Wing Commander
SirTech: Wizardry
SSI: Reach for the Stars

Computer Games / Interactive Adventure / Fiction
Infocom: Planetfall
LucasArts: The Secret of Monkey Island, Indiana Jones and the Fate of Atlantis, Sam 'n Max Hit The Road

Role Playing Games
Game Designers Workshop: *Traveller*
R. Talsorian: *Cyberpunk*

Network Resources
Jonathan P. Leech, *Space Frequently Asked Questions List*
Jennifer Smith, *ALT.GAMES.MUD Frequently Asked Questions List*
USENET *rec.games.programmer*
USENET *comp.multimedia*
USENET *alt.games.mud.**
USENET *sci.space.**
FTP *ftp.gmd.de Interactive Fiction Archive*
MUD source
x2ftp.oulu.fi:/pub/msdos/programming/gamsrc/abermud.zip
Graham Nelson, *Craft of Adventure.* Part of the Inform manual.Usenet.

Computer Software
Adobe *Illustrator 5.5*
Adobe *PhotoShop 3.0*
Autodesk *3D Studio R3/R4*
Corel *CorelDraw 5.0*
Equilibrium Technologis *Debabelizer 1.55*
Fauve *Matisse 1.2*
Macromedia *Director 4.0 for Macintosh*
Macromedia *Director 4.0 for Windows*
Quark *XPress 3.1*
Symantec *Think C++ 7.0*

APPENDIX A

Map Object Storage

This parent script contains all the ncessary data structures and handlers to support a 2D tile map display.

It relies on initialization of an X-Object of type "gScroller", which provides the extended scrolling display capabilities.

```
-- MapObj is a parent script (class) for manipulating
-- map structure, which are 2D grids (implemented as
-- nested lists) containing a single value representing
-- a particular tile.

-- PROPERTY VARIABLES
-- pMapArray is the 2D array representing the tile data.
-- pMapSizeX and pMapSizeY are the dimensions of the map.
-- pCenterX and pCenterY designates the "center tile" displayed
-- pBaseTile is the base cast number for determining tile #s
-- pInk is the "ink sprite" with trails on.
-- sprites pInk -1 and pInk -2 are used for char and background
-- storage...
-- pScroller points to my Scroller Xobject

property pDoldTX, pDoldTY
property pIoldX, pIoldY
property pFileName
property pMapArray, pMapSizeX, pMapSizeY
property pObjArray, pEnvArray
property pCenterX, pCenterY
property pBaseTile, pBaseThing, pBaseItem, pBaseEnv, pInk
property pScroller

-- These are used for figuring out how many tiles to draw
-- when refreshing or redrawing the screen. pTileOffset
-- determines the offset from the corner to draw...

property pTileL, pTileR, pTileT, pTileB
property pXstart, pYstart
property pTileOffsetX, pTileOffsetY

-- BIRTH HANDLER
-- Initialize the properties for this map:
-- sizeX, sizeY, and set all the tiles to 0 (null)
--
-- There is a minimum size...15 x 15 (arbitrary)
-- this fills the design screen nicely, thank you.

on birth me, sizeX, sizeY

  -- initialize the undo buffer
  set pUndoList = []

  -- initialize the default "ink sprite"
  set pInk = 48
  puppetsprite pInk -1, true       -- the character sprite
  puppetsprite pInk -2, true       -- the item sprite
  puppetsprite pInk -3, true       -- the background sprite
  --
  set pTileOffsetX = 0
  set pTileOffsetY = 0

  -- make sure it's at least 11 x 11
  if sizeX > 11 then
    set pMapSizeX = sizeX
  else
    set pMapSizeX = 11
  end if

  if sizeY > 11 then
    set pMapSizeY = sizeY
  else
    set pMapSizeY = 11
  end if

  -- center the map around the middle of the map
  set pCenterX = pMapSizeX / 2 + 1
  set pCenterY = pMapSizeY / 2 + 1

  set pBaseTile = the number of cast "BaseTile" + 1
  set pBaseThing = the number of cast "BaseThing" + 1
  set pBaseItem = the number of cast "BaseItem" + 1
  set pBaseEnv = the number of cast "BaseEnv" + 1


  -- initialize map array for instance
```

```
  -- row-column order!

  set pMapArray=[]
  set pObjArray=[]
  set pEnvArray=[]

  set temp = []

  repeat with i = 1 to pMapSizeX
    add(temp,0)
  end repeat

  repeat with i = 1 to pMapSizeY
    add(pMapArray,value(string(temp)))
    add(pObjArray,value(string(temp)))
    add(pEnvArray,value(string(temp)))
  end repeat

  set pDoldTX = -1
  set pDoldTY = -1
  set pIoldX = -1
  set pIoldY = -1

  return me
end birth

-- DISPLAY SUPPORT ACCESS METHODS
-- These are used when drawing the map to the screen.

-- define which scroll xObject will be used in scrolling
on setScroller me, xobj
  if objectP(xobj) then
    set pScroller = xobj
  end if
end

-- define the sprite to be used as the "brush" for updating
on setInkSprite me,spr
  set pInk = spr
end

-- define the offset for da map for when you draw...

on setTileOffset me,x,y
  set pTileOffsetX = x
  set pTileOffsetY = y
end

-- MAP MANIPULATION UTILITIES
-- utility methods for getting and setting particular tiles
-- in the map!

-- define the center tile to be displayed in the map
on goTile me,x,y
  if (x > 0) and (x <= pMapSizeX) and ¬
      (y > 0) and (y <= pMapSizeY) and ¬
      (getEnv(me,x,y)<>1) then
    set pCenterX = x
    set pCenterY = y
    set r = 1
  else
    set r = 0
  end if
  suUpdatePos me
  return r
end


-- set the tile/item/env in absolute map coordinates
on setTile me,tile,tx,ty
  if (tx>0) and (tx<=pMapSizeX) and (ty>0) and (ty<=pMapSizeY) then
    setAt(getAt(pMapArray,ty), tx, tile)
  else
    beep
    put "setTile: tile index out of range!"
  end if
end
```

```
on setItem me, thing, tx, ty
  if (tx>0) and (tx<=pMapSizeX) and (ty>0) and (ty<=pMapSizeY) then
    setAt(getAt(pObjArray,ty), tx, thing)
  end if
end

on setEnv me, env, tx, ty
  if (tx>0) and (tx<=pMapSizeX) and (ty>0) and (ty<=pMapSizeY) then
    setAt(getAt(pEnvArray,ty), tx, env)
  else
    beep
    put "setEnv: tile index out of range!"
  end if
end

-- retrieve the tile/Item/Env in absolute map coordinates
on getTile me,tx,ty
  if (tx>0) and (tx<=pMapSizeX) and (ty>0) and (ty<=pMapSizeY) then
    return getAt(getAt(pMapArray,ty),tx)
  else
    return -1
  end if
end

on getItem me, tx,ty
  if (tx>0) and (tx<=pMapSizeX) and (ty>0) and (ty<=pMapSizeY) then
    return getAt(getAt(pObjArray,ty),tx)
  else
    return -1
  end if
end

on getEnv me, tx,ty
  if (tx>0) and (tx<=pMapSizeX) and (ty>0) and (ty<=pMapSizeY) then
    return getAt(getAt(pEnvArray,ty),tx)
  else
    return -1
  end if
end


-- set the tile/item/env in the map, calculated from screen click
on setScreenTile me,tile,x,y
  set tx = (x-6)+pCenterX
  set ty = (y-6)+pCenterY
  setTile(me,tile,tx,ty)
end

on setScreenItem me,thing,x,y
  set tx = (x-6)+pCenterX
  set ty = (y-6)+pCenterY
  setItem(me,thing,tx,ty)
end

on setScreenEnv me,env,x,y
  set tx = (x-6)+pCenterX
  set ty = (y-6)+pCenterY
  setEnv(me,env,tx,ty)
end

-- get the tile/item/env in the map, calculated from screen click
on getScreenTile me,x,y
  set tx = (x-6)+pCenterX
  set ty = (y-6)+pCenterY
  return getTile(me,tx,ty)
end

-- get the tile that represents a particular fixed thing
-- (not an item)
-- which can't move, but operates nonetheless.
on getScreenThing me,x,y
  set t=getScreenTile(me,x,y)
  set t = t - (pBaseThing - pBaseTile)
  if t < 0 then
    return 0
  else
```

```
    return t
  end if

on getScreenItem me,x,y
  set tx = (x-6)+pCenterX
  set ty = (y-6)+pCenterY
  return getItem(me,tx,ty)
end

on getScreenEnv me,x,y
  set tx = (x-6)+pCenterX
  set ty = (y-6)+pCenterY
  return getEnv(me,tx,ty)
end



-- FILE I/O UTILITIES for loading and saving data

on GetFileName me
  return pFileName
end

-- Save the file as a series of lists in STRING
-- format. First two strings are WIDTH and HEIGHT,
-- followed by HEIGHT number of arrays of WIDTH size

on SaveFile me
  put FileIO(mNew,"?write","") into fo
  if not objectP(fo) then
    beep
    put "error in creating file object"
    exit
  else
    set err = fo(mSetFinderInfo,"TEXT","R*ch")

    set nL = count(pMapArray)

    set nW = count(getAt(pMapArray,1))
    set err = fo(mWriteString,string(nL))
    set err = fo(mWriteString,RETURN)
    set err = fo(mWriteString,string(nW))
    set err = fo(mWriteString,RETURN)
    repeat with i = 1 to nL
      set err = fo(mWriteString,string(getAt(pMapArray,i)))
      set err = fo(mWriteString,RETURN)
      set err = fo(mWriteString,string(getAt(pObjArray,i)))
      set err = fo(mWriteString,RETURN)
      set err = fo(mWriteString,string(getAt(pEnvArray,i)))
      set err = fo(mWriteString,RETURN)
    end repeat
  end if
  set err = fo(mDispose)
end

on LoadFile me, fName
  put FileIO(mNew,"read",fName) into fo
  if not objectP(fo) then
    beep
    PrintError "Can't find map file...end program."
    stopmovie
  else
    set the itemDelimiter = ":"
    set pFileName = the last item of fName
    set the itemDelimiter = ","

    set pMapSizeY=value(fo(mReadLine))
    set pMapSizeX=value(fo(mReadLine))
    set pMapArray=[]
    set pObjArray=[]
    set pEnvArray=[]

    set tlic = 0
    repeat with i = 1 to pMapSizeY
      set tlic = tlic + 1
      if tlic > 2 then
        set tlic = 0
```

```
        ToggleLoadIndicator
      end if
      add (pMapArray, value(string(fo(mReadLine))))
      add (pObjArray, value(string(fo(mReadLine))))
      add (pEnvArray, value(string(fo(mReadLine))))
    end repeat
    HideLoadIndicator
    set pCenterX = pMapSizeX / 2 + 1
    set pCenterY = pMapSizeY / 2 + 1
  end if
  set err = fo(mDispose)
end


-- MAP DRAWING METHODS

-- boundDrawMap is supposed to determine the beginnning and ending
-- tiles used in drawing the map. Also computes start point.
on boundDrawMap me
  set pTileL = pCenterX - 5
  set pTileR = pCenterX + 5
  set pXstart = 0

  if pTileL < 1 then
    set pXstart = (1+abs(pTileL)) * 32
    set pTileL = 1
  end if

  set pXstart = pXstart + pTileOffsetX

  if pTileR > pMapSizeX then
    set pTileR = pMapSizeX
  end if

  set pTileT = pCenterY - 5
  set pTileB = pCenterY + 5
  set pYstart = 0

  if pTileT < 1 then
    set pYstart = (1+abs(pTileT)) * 32
    set pTileT = 1
  end if

  set pYstart = pYstart + pTileOffsetY

  if pTileB > pMapSizeY then
    set pTileB = pMapSizeY
  end if
  portCMD "POS" && pFileName & "," & pCenterX & "," & pCenterY
end

-- draw the map, based on the bounds and center tile defined
-- above!
on drawMap me
  -- x,y is coordinate of tile to draw in the center
  -- of the screen

  boundDrawMap(me)

  -- draw the map
  set sx = pXstart
  set sy = pYstart

  eraseMapDisplay -- external call

  repeat with j = pTileT to pTileB
    set the locV of sprite pInk = sy
    repeat with i = pTileL to pTileR
```

```
      set the locH of sprite pInk = sx
      set the castnum of sprite pInk = pBaseTile + getTile(me,i,j)
      updatestage
      set the ink of sprite pInk = 36
      set the castnum of sprite pInk = pBaseItem + getItem(me,i,j)
      updatestage
      set the ink of sprite pInk = 0
      set sx = sx + 32
    end repeat
    set sx = pXstart
    set sy = sy + 32
  end repeat

  -- update the background sprite
  drawCenterTile(me)
  hidechar
  showchar
  -- hide the ink tile so it doesn't leave a mess
  set the locH of sprite pInk = -64
end

-- draw just the top row of tiles, if they are visible
on drawTopTiles me

  boundDrawMap(me)

  set the locV of sprite pInk = pTileOffsetY

  -- shouldn't draw tiles if pCenterY is less than 8

  if (pCenterY>5) then
    -- draw the map
    set sx = pXstart
    set j = pTileT
    repeat with i = pTileL to pTileR
      set the locH of sprite pInk = sx
      set the castnum of sprite pInk = pBaseTile + getTile(me,i,j)
      updatestage
      set the ink of sprite pInk = 36
      set the castnum of sprite pInk = pBaseItem + getItem(me,i,j)
      updatestage
      set the ink of sprite pInk = 0
      set sx = sx + 32
    end repeat
  end if
  set the locH of sprite pInk = -64
end

-- draw the bottom row of tiles, if they are visible
on drawBottomTiles me

  BoundDrawMap(me)

  set the locV of sprite pInk = 10 * 32 + pTileOffsetY

  -- shouldn't draw tiles if pCenterY is within 7 of pMapSizeY
  if (pMapSizeY-pCenterY) > 4 then
    -- draw the map
    set sx = pXstart
    set j = pTileB
    repeat with i = pTileL to pTileR
      set the locH of sprite pInk = sx
      set the castnum of sprite pInk = pBaseTile + getTile(me,i,j)
      updatestage
      set the ink of sprite pInk = 36
      set the castnum of sprite pInk = pBaseItem + getItem(me,i,j)
      updatestage
      set the ink of sprite pInk = 0
      set sx = sx + 32
    end repeat
  end if
  set the locH of sprite pInk = -64
end

-- draw the left column of tiles, if visible
on drawLeftTiles me
```

```
  BoundDrawMap(me)
  set the locH of sprite pInk = pTileOffsetX

  -- shouldn't draw if pCenterX is less than 8
  if pCenterX > 5 then
    -- draw the map
    set sy = pYstart
    set i = pTileL
    repeat with j = pTileT to pTileB
      set the locV of sprite pInk = sy
      set the castnum of sprite pInk = pBaseTile + getTile(me,i,j)
      updatestage
      set the ink of sprite pInk = 36
      set the castnum of sprite pInk = pBaseItem + getItem(me,i,j)
      updatestage
      set the ink of sprite pInk = 0
      set sy = sy + 32
    end repeat
  end if
  set the locH of sprite pInk = -64
end

-- draw the right column of tiles, if visible
on drawRightTiles me

  BoundDrawMap(me)
  set the locH of sprite pInk = 10 * 32 + pTileOffsetX

  if (pMapSizeX-pCenterX) > 4 then
    -- draw the map
    set sy = pYstart
    set i = pTileR
    repeat with j = pTileT to pTileB
      set the locV of sprite pInk = sy
      set the castnum of sprite pInk = pBaseTile + getTile(me,i,j)
      updatestage
      set the ink of sprite pInk = 36
      set the castnum of sprite pInk = pBaseItem + getItem(me,i,j)
      updatestage
      set the ink of sprite pInk = 0
      set sy = sy + 32
    end repeat
  end if
  set the locH of sprite pInk = -64
end

-- redraw the tile at screen tile coordinates
on drawScreenTile me, x, y
  set tx = (x-6)+pCenterX
  set ty = (y-6)+pCenterY
  if (tx>0) and (tx<=pMapSizeX) and (ty>0) and (ty<=pMapSizeY) then
    set the locH of sprite pInk = (x-1) * 32 + pTileOffsetX
    set the locV of sprite pInk = (y-1) * 32+ pTileOffsetY
    set the castnum of sprite pInk = pBaseTile + getTile(me,tx,ty)
    updatestage
  end if
end

on drawScreenItem me,x,y
  set tx = (x-6)+pCenterX
  set ty = (y-6)+pCenterY
  if (tx>0) and (tx<=pMapSizeX) and (ty>0) and (ty<=pMapSizeY) then
    set the castnum of sprite pInk = pBaseItem + getItem(me,tx,ty)
    set the locH of sprite pInk = (x-1) * 32 + pTileOffsetX
    set the locV of sprite pInk = (y-1) * 32 + pTileOffsetY
    set the ink of sprite pInk = 36
    updatestage
    set the ink of sprite pInk = 0
  end if
end

on eraseScreenGuy me
  setItem me,0,pioldx,pioldy
  if ( abs(pDoldTX-pCenterX)<6) and (abs(pDoldTY-pCenterY)<6) then
    drawScreenTile me,pDoldTX-pCenterX+6,pDoldTY-pCenterY+6
```

```
  end if
end

-- given global tile coords...
on drawScreenGuy me, tx, ty
  setItem me,0,pioldx,pioldy
  setitem me,1,tx,ty
  set pioldx=tx
  set pioldy=ty

  -- local tile coordinates are x and y.
  -- got to check if this is on the screen
  if ( abs(pDoldTX-pCenterX)<6) and (abs(pDoldTY-pCenterY)<6) then
    drawScreenTile me,pDoldTX-pCenterX+6,pDoldTY-pCenterY+6
  else
    nothing
  end if
  set pDoldTX = tx
  set pDoldTY = ty
  set x = tx - pCenterX + 6 -- get screen cordinates
  set y = ty - pCenterY + 6
  if (x<12) and (x>0) and (y<12) and (y>0) then
    drawscreenitem me,x,y
    drawCenterTile me
  else
    nothing
  end if
end


-- draw the center tile on the screen
on drawCenterTile me
  set the castnum of sprite (pInk - 3) = pBaseTile +
getTile(me,pCenterX,pCenterY)
  set the castnum of sprite (pInk - 2) = pBaseItem +
getItem(me,pCenterX,pCenterY)
  updatestage
end

on ShowChar
  set the visible of sprite pInk -1 = TRUE
  updatestage
end

on HideChar
  set the visible of sprite pink -1 = FALSE
  updatestage
end

-- scroll the screen down
on goNorth me
  if goTile(me,pCenterX,pCenterY-1) then
    HideChar
    pScroller(mScroll,0,32)
    drawCenterTile(me)
    ShowChar
    DrawTopTiles(me)
  else
    puppetsound 0
    updatestage
    puppetsound "tink"
    updatestage
  end if
end

-- scroll the screen up
on goSouth me
  if goTile(me,pCenterX,pCenterY+1) then
    HideChar
    pScroller(mScroll,0,-32)
    drawCenterTile(me)
    ShowChar
    DrawBottomTiles(me)
  else
    puppetsound 0
    updatestage
    puppetsound "tink"
```

```
    updatestage
  end if
end

-- scroll the screen left
on goEast me
  if goTile(me,pCenterX+1,pCenterY) then
    HideChar
    pScroller(mScroll,-32,0)
    drawCenterTile(me)
    ShowChar
    DrawRightTiles(me)
  else
    puppetsound 0
    updatestage
    puppetsound "tink"
    updatestage
  end if
end

-- scroll the screen right
on goWest me
  if goTile(me,pCenterX-1,pCenterY) then
    HideChar
    pScroller(mScroll,32,0)
    drawCenterTile(me)
    ShowChar
    DrawLeftTiles(me)
  else
    puppetsound 0
    updatestage
    puppetsound "tink"
    updatestage
  end if
end
```

Map Utilities

This script determines which tile is being clicked by the user.

Additional calculation is required by the Map Object scripts. See that script for more information.

```
-- MAP UTILITIES
-- routines related to clicking on the tile map
-- on the screen. To manipulate the map data and
-- current location, see the MapObj parent script.

-- note: the upper left origin of the tile display is
-- located at 32,13

on clickedScreenX
  set x = the mouseH - 32
  if x>479 then
    set r = 0
  else
    set r = x / 32 + 1
  end if
  return r
end

on clickedScreenY
  set y = the mouseV - 13
  if y>479 then
    set r = 0
  else
    set r = y / 32 + 1
  end if
  return r
end

on eraseMapDisplay
  set the ink of sprite 1 = 8
  updatestage
  set the ink of sprite 1 = 0
end
```

Tile Grabber

This script uses the gScroller X-Object to capture, cut and paste new tiles from a PICT template into the cast of a Director movie.

```
-- MakeTilesFromPict

-- Pict is stored in cast member 2 when calling
-- MakePictFromTiles. Tiles are made from the current
-- screen display when using MakeTilesFromPict

-- 5 columns across of tiles. Each column is 3 tiles
-- wide, with the fourth tile used as a buffer space.
-- Like this:
---
-- 123 ### ### ### ###
-- 456 ### ### ### ###
-- ### ### ### ### ###
--
-- Scan from 123456, as shown above. When bottom is
-- reached, the next column is scanned from the top.
--
-- The TILE PICT is loaded ALWAYS into the cast number
-- specified by the variable PICTCAST.
--
-- Newly-clipped sprites are put in the cast number
-- specified in variable BASECAST. Will load 240 tiles,
-- so make sure there's enough space!

on makeTilesFromPict PictName

  global gFloyd

  set BaseCast = the number of cast "BaseTile"
  if BaseCast < 100 then
    beep
    put "Error Finding BaseCast!"
    halt
  else
    set BaseCast = BaseCast + 1
  end if

  set PictCast = 2

  ImportFileInto cast PictCast, PictName
  set the regpoint of cast PictCast = point (0,0)

  set counter = 0

  repeat with i = 0 to 2

    repeat with y = 0 to 14
      repeat with x = 0 to 2
        set x1 = (i * 128) + (x*32)
        set y1 = y * 32
        gFloyd(mSetRect,x1,y1,x1+32,y1+32)
        gFloyd(mCopy)
        pasteClipboardInto cast (basecast + counter)
        set the regpoint of cast (basecast + counter) = point (0,0)
        set counter = counter + 1
      end repeat
    end repeat
  end repeat
end
```

Player Link Object

This parent script handles the tricky business of keeping track of two people in the same world.

The first thing the game does is create an instance of the script to keep track of the player. It checks to see if there is another player on another computer. If one is found, another uinstance of this script is created.

There are two tasks that this script performs. The first task is to tell other players when a particular character moves. The other task is to respond to other players in an intelligent way.

```
-- parent script to handle serial-serial link!

property pName
property pMapX, pMapY
property pMapName
property pColor
property pAlive

-- first thing, send a "hello" out to the world.

on birth me
  set pName = "noname"
  set pMapName = ""
  set pMapX = -1
  set pMapY = -1
  set pAlive = FALSE
  return me

end

on updateMyPos me,gmap
  set pMapName = the pFileName of gmap
  set pMapX = the pCenterX of gmap
  set pMapY = the pCenterY of gmap
end


on updateYourPos me,mapobj
  if pMapName = the pFileName of mapobj then
    drawscreenguy(mapobj,pMapX,pMapY)
  end if
end

on resurrect me
  birth me
end

on isAlive me
  return pAlive
end



-- send out the Hello...

on sendHI me
  portCMD "HI" && pName & "," & pMapName & "," & pMapX & "," & pMapY
end

on waitHiAck me,aPort

  set wLen = 15
  set wTimeOut = 3 * 60
  set wPC = 0

  set ts = the timer
  set te = ts + wlen
  set ACK = 0
  repeat while not Ack
    if aPort(mReadCount) > 0 then
      set aString = aString & aPort(mReadString)

      set lchar = the last char of aString
      if lchar = RETURN then
        set cmd = word 1 of aString
        delete word 1 of aString
        if CMD = "NAME" then
          getName0 me, aString
          set wPC = wPC + 1
          set aString = ""
        else if CMD = "POS" then
          getPos me, aString
          set wPC = wPC + 2
          set aString = ""
        end if

        if (wPC <> 1) and (wPC <> 3) then
```

```
          set wPC = 0
          printError "wPC in hiAck incorrect"
        end if

        if wPC = 3 then
          set ACK = 2
        end if
      else -- if last char not RETURN
      end if
    end if
    if the timer > te then
      ToggleLoadIndicator
      set te = the timer + wlen
    else if (the timer - ts) > wTimeOut then
      set ACK = 1
    end if
  end repeat
  HideLoadIndicator
  return ACK - 1
  -- 1 success, 0 timeout
end


-- dudeWelcomeAck grabs info about newcomer,
-- and sends info about ourselves

on getHI me, str
  global gMap
  set pName = item 1 of str
  set pMapName = item 2 of str
  set pMapX = value(item 3 of str)
  set pMapY = value(item 4 of str)
  printStatus doQUote(pName) && "has arrived"
  set pAlive = TRUE
  drawScreenGuy(gMap, pMapX, pMapY)
end

on sendName me
  portCMD "NAME" && pName
end

on changeName me, str
  set pName = str
  PrintStatus "You are now known as" && doQUote(pName)
  sendName me
end

on getName me, str
  set oldname = pName
  getName0 me,str
  printStatus doQUote(oldname) && "is now called" && doQuote(pName)
end

on getName0 me,str
  if str contains RETURN then
    set str = chars(str,1,length(str)-1)
  end if
  set pName =  str
end

on setPosFromMap me,mapObj
  set pMapName = the pFileName of MapObj
  set pMapX = the pCenterX of mapObj
  set pMapY = the pCenterY of mapObj
end

on sendPos me
  portCMD "POS" && pMapName & "," & pMapX & "," & pMapY
end

on getPos me, str
  -- update position of other player
  global gMap
  set pMapName = item 1 of str
  set pMapX = value(item 2 of str)
  set pMapY = value(item 3 of str)
  if pMapName = the pFileName of gMap then
```

```
    drawScreenGuy(gMap,pMapX,pMapY)
  else
    eraseScreenGuy(gMap)
  end if
end

on getBye me,str -- contains leaving message
  global gMap
  printStatus pName && "has left the game."
  resurrect
  eraseScreenGuy(gMap)
  set pAlive = FALSE

end

on sendBye me
  portCMD "BYE" && "testbye"
end

on doAction me, str
  portCMD "SYS" && pName && str
  printStatus pName && str
end
```

Serial Port Utilities

This script sets up the Macintosh's serial ports for communication. It also handles the low-level serial port communication, making sure that messages are completely received and processed by the appropriate handlers in the Player Link Object.

```
on initPort
  global gPort

  -- make sure there's no other instance
  if objectP(gPort) then gPort(mDispose)

  -- 0 is modem port, 1 is printer port
  put SerialPort(mNew,0) into gPort

  -- Set 38.4kbps, 1 stop bit, no parity
  gPort(mSetup,38400,10,0)

  -- Set up hardware handshaking (CTS/DTR) for input (0) and output (1)
  gPort(mHShakeChan,0, 2+8, 17)
  gPort(mHShakeChan,1, 2+8, 17)

  -- see page 315 of "Using Lingo" for more serial port commands

end

on killPort
  global gPort

  if objectP(gPort) then
    gPort(mDispose)
  end if
end

-- I     mGetPortNum --> the port.
-- IS    mWriteString, string --Writes out a string of chars.
-- II    mWriteChar, charNum --Writes a single character.
-- S     mReadString --> the contents of the input buffer.
-- I     mReadChar --> a single character.
-- I     mReadCount --> the number of characters in the input buffer.
-- X     mReadFlush --Clears out all the input characters.

-- Command Structure:
-- <CMD> <STUFF>
--
-- CMD is a single word. The first word of the sent string is
-- always assumed to be the command string. Unrecognized
-- commands are ignored and flushed.
--
-- The STUFF following the command is the rest of the string,
--the format of which depends on what command.
--

on portIncoming
  global gPort, gInput
  if gPort(mReadCount) then
    set gInput = gInput & gPort(mReadString)
    set lchar = the last char of gInput
    if lchar = RETURN then
      portparseCmd
    end if
  end if
end

on portParseCmd
  global gPort, gInput
  global gMe, gDude

  set cmd = word 1 of gInput
  delete word 1 of gInput
  --  set gInput = chars(gInput,1,(length(gInput)-1))
  delete the last char of gInput

  if cmd = "TALK" then
    PrintYou gInput
  else if cmd = "POS" then
    getPos(gDude,gInput)
  else if cmd = "HI" then
    getHi(gDude,gInput)
    sendName(gMe)
    sendPos(gMe)
  else if cmd = "BYE" then
    getBye(gDude,the pName of gMe & " has left the game")
```

```
    -- stopmovie
  else if cmd = "SYS" then
    PrintStatus gInput
  else if cmd = "NAME" then
    getName(gDude,gInput)
  else if cmd = "OTVGO" then
    PrintDivider
    PrintStatus the pName of gDude && "has launched the OTV!"
    PrintStatus "You are now on your way to the moon."
    PrintStatus "*NOTE* This is the end of the interactive demo."
    PrintDivider
    go "OTVGo"
  else
    nothing
  end if
  set gInput = ""
end

on portCmd str
  global gPort
  waitTicks 7
  gPort(mWriteString,str&RETURN)
end

on suUpdatePos mapobj
  global gMe
  updateMyPos(gMe,mapobj)
  sendPos(gMe)
end
```

Game Response Utilities

This script provides feedback to the player when they perform some action. The basic Input/Output utilites are in this script, as are the "loading game" toggle indicator and random responses to player actions.

```
on ParseCommand str
  global gMe

  if char 1 of str = "/" then
    set myCmd = word 1 of str
    delete char 1 of myCmd
    delete word 1 of str

    if myCmd = "scream" then
      printStatus "In space, no one can hear ya..."
    else if myCmd = "jump" then
      printStatus "Don't jump in zero gee!"
    else if myCmd = "help" then
      printStatus "Commands so far are name, me, where, scream, and
jump."
    else if (myCmd = "where") or (myCmd = "where?") then
      PrintWhereAmI
    else if myCmd = "name" then
      if length(str) >0 then
        changeName(gMe,str)
      else
        printStatus "Example use of the /name command: /name Joe"
      end if
    else if myCmd = "me" then
      if length(str) >0 then
        doAction(gMe,str)
      else
        printStatus "Example use of the /me command: /me sings."
      end if
    else if myCmd = "quit" then
      stopmovie
    else  if myCmd = "" then
      PrintStatus "The / and Command are one word."
    else
      printStatus "unrecognized command"
    end if
  else
    PrintMe str
  end if
end

on doQuote str
  return QUOTE & str & QUOTE
end

on printDivider
  printString "---------------------------------"
end

on PrintWhereAmI
  global gMap

  set str = GetFileName(gMap)
  if str = "plane-1.map" then
    PrintStatus "You're on the upper deck of the Clipper."
  else if str = "plane-2.map" then
    PrintStatus "You're on the lower deck of the Clipperr."
  else if str = "station.map" then
    PrintStatus "You're on-station the Charleston."
  else if str = "otv.map" then
    PrintStatus "You're on an H-OTV bound for Luna City."
  else
    PrintStatus "What is this place? Argh!"
  end if
end

on PrintString str
  put str & RETURN after field "Scrolly"
  if (length(str)>45) and (the number of lines of field "Scrolly" >4)
then
    put "" after field "Scrolly"
  end if
end

on PrintError str
  PrintString "*** ERROR ***" && str
end
```

```
on PrintStatus str
  puppetsound 0
  updatestage
  puppetsound "telclick"
  updatestage
  PrintString "[]" && str

  --  set the forecolor of field "Scrolly" = 5
  --  waitTicks 3
  --  set the forecolor of field "Scrolly" = 83
end

on PrintMe str
  PrintString QUOTE & str & QUOTE
  portCMD "TALK" && str
end

on PrintYou str
  global gDude

  PrintString the pName of gDude & ": " & QUOTE & str & QUOTE

  set the forecolor of field "Scrolly" = 114
  waitTicks 3
  set the forecolor of field "Scrolly" = 83

end

on ToggleLoadIndicator
  set flag = the visible of sprite 9
  set the visible of sprite 9 = 1 - flag
  updatestage
end

on HideLoadIndicator
  set the visible of sprite 9 = true
  updatestage
end

on ShowLoadIndicator
  puppetsound 0
  updatestage
  puppetsound "beephi"
  set the visible of sprite 9 = false
  updatestage
end


on setcastname spr, str
  set the castnum of sprite spr = the number of cast str
end

on waitTicks numticks
  set tend = the timer + numticks
  repeat while the timer < tend
  end repeat
end

on hidecursor
  cursor 200
end

on showcursor
  cursor -1
end

on hideInks
  repeat with x = 0 to 3
    set the visible of sprite 48 - x = false
  end repeat
  updatestage
end

on showInks
  repeat with x = 0 to 3
    set the visible of sprite 48 - x = true
  end repeat
  updatestage
end
```

```
on randomLookAction
  global gMe, gDude

  printStatus "You look at" && the pName of gDude
  set rList = ¬
    ["glances furtively in your direction.",¬
     "clicks on your icon.",¬
      "looks right at you.",¬
      "seems to be looking at you...",¬
      "looks in your direction."]

  set r = count(rList)

  set str = the pName of gMe && getAt(rList,random(r))
  return str
end
```

APPENDIX B

Planetary Data (from *Fundamentals of Physics 2nd Edition*, Halliday, Resnick, John Wiley & Sons, NY)

	Mercury	Venus	Earth	Mars	Jupiter	Saturn	Uranus	Neptune	
mean distance from sun (10^6km)		57.9	108	150	228	778	1430	2870	4500
period of revolution (years)	.241	.615	1.00	1.88	11.9	29.5	84.0	165	
orbital speed (km/s)	47.9	35.0	29.8	24.1	13.1	9.64	6.81	5.43	
diameter (km)	4880	12100	12800	6790	143000	120000	51800	49500	
gravity (m/s^2)	3.78	8.60	9.78	3.72	22.9	9.05	7.77	11.0	
escape velocity (km/s)	4.3	10.3	11.2	5.0	59.5	35.6	21.2	23.6	

Earth Data (from Fundamentals of Physics 2nd Edition, Halliday, Resnick, John Wiley & Sons, NY)

	Earth	Moon
Mean radius (km)	6370	1740
Surface gravity (m/s^2)	9.81	1.67
Escape velocity (km/s)	11.2	2.38

Transit Times
For 2G burn at 90 seconds (similar to lunar excursion module), it takes about 5 days to get to the moon. For 3G burn full, transit time is 1.5 hours. For 1G burn full, it's about 3 hours. There's a range of expenses associated with the time!

Distance from Moon to L4 or L5 (60 degrees forward and back respectively), the distance is 425,000km.

Equation: T = ((2AD)^0.5) / A no stopping
Equation: T = 2(AD)^0.5 / A stopping at end of distance

Orbit Facts (source: Space stations and platforms)

< 160km short lifetime
> 1000km severe Van Allen radiation belt effects
> 20000km past the belts, but you're now pelted with cosmic radiation

600km is pretty good, though space station can be as low as 200km.

orbital v = (u/r)^0.5, where u/r^2 = geopotential = 398,601km^3/sec^2
period of rotation = 2pi(r^3/u)^0.5

@200km, orbital velocity is 7.7 km/s
@1000km, orbital velocity 7.2 km/s

Orbits the earth every 1.5 hours.

Repetitious Orbits
A track can be chosen for low earth orbit that continually passes over the same areas over a period of days.

Sun Synchronous Orbits
These are always fully bathed in sunlight all the time.

Orbit Degradation due to atmospheric braking
The change in attitude is 100s of meters per day. Drag thrusted needed to overcome this is very small, however...on the order of 1/50th of a pound.

Docking
This is when two craft collide together at slow speed (around 0.3m/s) to engage a locking ring.

Berthing
This is a mechanical coupling that draws the craft together...join and tug. Accuracy required is about 15 cm position, within 0.1 m/s velocity.

Gravity Gradient Stable versus Rotational
Since the force of gravity is different for different altitudes, an irregularly shaped mass will tend to orient towards the earth. This is called gravity gradient stable. A rotating design, on the other hand, poses problems for stability in orbit due to more complex control requirements.

Radiation Effects on Spacecraft Materials
Ultraviolet light that is usually filtered out by the atmosphere affects certain materials. Plastics, paints, adhesives, and glass are all susceptable. This can be eliminated by the use of special UV blocking paints, foil wrappings, or metalized films.

Radiated Power per Square Meter
Sunlight incident on a square meter is approximately 1353 watts per square meter, up to 1425. The earth's albedo (reflected radiation) can be as much as half of this figure. This is the primary heat input to the spacecraft. Conductive fin coolers are required to radiate excess heat...this places demands on the physical configuration of the station / craft.

Ionizing Radiation
From 200km to 2000km, there is a significant amount of trapped radiation in the form of ionized atmospheric gasses. These are the Van Allen belts. The earth's magnetic field traps the ionizing radiation, and it particularly strong of the South Atlantic (the "South Atlantic Anomoly" is due to some weirdness with the Earth's magnetic field). For a craft in low earth orbit, the craft WILL pass through this area, which makes extravehicular activity (EVA) unfeasible. There is, however, a 10 hour period of consecutive orbits that do not have lots of radiation hazards outside the craft.

Spacecraft Charging
The sun's energy can cause electron charges to build up on exposed metal surfaces, causing voltage gradients across the vessel. These can discharge as arcs of electricity, which can cause damage to electronics and optics.

Cosmic Radiation
Above 2000km, the craft is free of ionizing radiation. However, the lack of atmosphere means that cosmic radiation becomes the most significant electomagnetic hazard. Increase in shielding will tend to increase secondary radiation resulting from high-speed collisions with the shield's atomic matrix, until a certain point is reached (on the order of 1000 gm/cm^2 of shielding).

Solar Flares
Outside of the earth's magnetic field, solor ionizing radiation is also a major concern. It consists of a stream of hydrogen nuclei with high energy potentials, accelerated by the sun's hefty electromagnetic fields. During solar flare activity, people caught in the open without adequate shielding will risk radiation poisoning. Significant solar flares require much shielding for protection, or some kind of abort mode to return to the earth. A solar flare can last from hours to days.

Radiation Exposure Limits
No real numbers cranked out yet.

Upper Atmosphere Composition
The upper atmosphere consists mostly of hydrogen and atomic (unbounded) oxygen. This is highly reactive as an oxidizer, and can cause severe erosion of materials. The stream of atomic oxygen bombards spacecrafts at around 7km/s (orbital velocity) in a fully-elastic collision. This can eat away from 365-4000 microns every year. The space shuttle exhibited signs of such erosion in their tiles.

Meteroids
A meteor is a meteroid that has fallen to earth. Meteroids, on the other hand, are usually small, fast, and largely invisible. There is a chance of collision of about 0.001 per 2000 square meters of material every year. (? not sure about number ?)

Man-Made Debris
In low earth orbit, man made debris is everywhere. There is a probability of 0.02 collisions per year with a 2000 square meter array. This rises to about 0.1 collisions/year at higher altitudes (different sources say different things).

Hull Construction
Protection by dual-wall or multi-wall shields and armor should provide adequate protection. Penetration could be significant amount (.1 cm????)

Electrical Considerations
Skylab uses 18kilowattsof electricity. A space station with 60 people on board could use up to 200kW, which provides ample power for materials processing and electricity. Provided by solar panels (about 10% efficient at gathering sunlight of 1300watts per square meter) or cassegrainian concentrators (parabolic reflector dishes concentrating sunlight on a GaAs cell).

Cooling
Radiation Fins, outplaced nuclear reactors, effective reflectors facing the sun, heat pipes for pumping away heat using liquid or liquid-metal convection systems. Special coatins that are selectively reflective at one energy and conductive at others are useful. Panels should be armored.

Humans generate about 136 watts of heat (400BTU/hour). 35% of this is in water vapor. During sleep, this figure is abou;half. During active exercise, 3 to 5 times as high.

Ventillation
Forced convection is required to make air circulate in the absence of gravity, otherwise unpleasant dead spots will form. Air flow rate of 7.5 meters/ minute are about right. 12 cm/sec is comfortable.

Atmosphere
Skylab had 5 pounds/square inch (psia) in a pure oxygen system. Shuttle uses a 2 gas oxygen + nitrogen system at 14.7 psia. Pressure differentials between space suits and shuttle cabin requires use of airlock to prevent aeroembolism ("the bends"). In the event of inadverdent decompression, emergency airlock should be usable as a hyperbaric (high pressure) system to prevent the bends.

Zero-G effects

After about 2 weeks without exercise in zero-g, people can not stand up when returned to Earth. 1-2 hours of exercise per day will maintain strength. Bone loss occurs about 1% per month in absense of gravity. Calcium is lost through urine and feces. 5% of astronauts experienced space sickness that did not pass for 2-4 days. Some threw up.

Exposure to Vacuum

People do not explode. Blood does not boil. People do not freeze, as vacuum is an insulator. If breath is held, lungs may burst. Ear drums may also burst. 1-2 minutes of exposure to vacuum can pass within permanent damage. Some swelling may occur. Past this, dehydration of mucous membranes occurs. Brain damage may occur due to oxygen starvation. Aeroembolism can occur. Radiation damage to skin also occurs.

APPENDIX C

A chunk of prose written to "feel out" the setting of the game.

The eastern launching zone starts at Cape Kennedy and extends for another 100 miles into the Atlantic. You have to get there a day early to go through the mandatory preflight physical, though they say it's better to get there a week *just in case* you have something wrong with you. It's a lie...either you're in shape for the flight or not. This hasn't stopped a small hotel industry from booming in the area, serving the newbie space travellers and their extended families who have come to watch.

There's really nothing to see anymore. In the old days, you got to sit on top of a giant firecracker that blew you into the sky in seconds, leaving a massive pillar of vapor to mark your passage into history. Now, we have these cool but somewhat bus-like high-altitude transfer vehicles, which are nothing more than modified jumbo HSTVs you fly every day from New York to Taipei in just under four hours. The only difference is the shuttle itself, slung somewhat precariously under the HSTV in some incredibly complex aerodynamic way...it's imaginatively called the "THSTV configuration". The aerospace guys are always bringing it up at their conferences, but I get the feeling that not all of them are congratulating themselves on the design. That should have me worried, but I attribute it to the losers of the THSTV competition.

I breeze in to the pre-launch compound about an hour before they seal the gates. There are a handful of people, maybe 20 or so, still lingering outside dragging out their tearful goodbyes. It's kind of eerie to see so many clean-cut family people in one place. It's like watching Nick at Night in 3D. Space travel makes the earthbound nervous. Those who were left behind will camp out in the luxurious observation terminal, anxiously nibbling on free hors d'oevres as they watch the THSTV trundle up the 5 mile-long runway. A lot of them wave (yes, I've been there for the free food). After the THSTV disapears from view, they can wait for two hours to make sure that the shuttle seperation went OK, high over the earth. Most of them then go home, by way of Disneyland.

In the compound, there is a lot of preflight checking out of bodies. There are mandatory lectures on how to cope with space sickness, how to relax without getting out of your chair, and how SAFE space travel is in the Silver Age of Spaceflight. There's also a convenient kiosk where you can fill out Space Accident insurance for $10, and it looks like most people will hedge their bets. The scientific types are hanging back, but the tourists are eagerly filling out the forms, not realizing that the insurance covers only the trip to the transfer station in low earth orbit. You can buy more insurance once you're stranded in space. I notice for the first time that the compound design is very much like the interior of the shuttle, except it's a lot bigger of course. There are always smiling attendents and cheery posters of space, with very small windows. Maybe this is to help the newbie travellers get used to the cramped confines of the shuttle, which probably is a good idea.

Everybody and every thing that is going on-shuttle gets weighed. Stuff that looks like it will get broken and cause trouble is weeded out. There is some grumbling by the passengers, but they'll have all night to get over it.

A light dinner is served...nothing gross or smelly that could cause nausea later in the flight. Just 16 hours to go to launch. I spot only one kid in the

line...that's a blessing. I think they should ban children from spaceflight, or at least have special flights that are adult-only. On a closer look, the kid looks very bright-eyed and enthusiastic, so he might be one of those *gonna be a scientist!* types. I stay out of his way.

We're woken up six hours before the flight. We are offered a tasty ìSpace Shakeî that we are required to drink. It contains anti-retching drugs, relaxants, and some powerful laxatives to compel us to go to the bathroom. The spaceflight is a long fast until we arrive on-station 18 hours from now.

Drained, we all start walking down the enclosed boarding dock to board the transfer bus that will take us to the shuttle. It too has no windows except for some tiny portholes, and we are separated from the driver. A well-aerobicized flight attendent boards with us, and talks pleasantly of the flight to come, hitting all twelve pre-flight safety tips in sixty seconds. None of us have bags, since they are disallowed in the passenger cabin. We do have pockets...everyone is wearing a NASA-approved jumpsuit, and NASA loves pockets. Some of us are wearing standard issue suits, but the tourists (and some scientists who get dressed by their wives) are wearing off-the-rack fashions from the Mall.

The bus slows, and the entire passenger container rises to the shuttle...no chance to look outside. The flight attendent smiles reassuringly, and we follow her single-file into the shuttle itself.

The shuttle compartment is a rounded 20 feet wide and 10 feet tall at the center. There is room for 30 people, but there are some emtpy seats. The seats are kind of narrow but snug, with plenty of leg and head room. We're being strapped in by the flight attendent. There will be a period of time when we can not release the straps, but we're not told this (makes people nervous). We are told, though, not to remove the straps for any reason until we reach orbit. Most people will obey this, except for the occasional stupid kid. Hence, the irremovable straps.

I'm getting hungry. Can't wait until we're on-station. My stomach growls loudly to my embarassment, and the person to my side looks at me and smiles *me too.*

After the clacking of the strap-in period ends, the shuttle is silent except for an impossibly-even electric humming noise. Is it synthesized or real? We can feel some unidentifiable thumps and thuds as the THSTV is prepared for launch. This never takes long, thankfully...the waiting drives some people nuts.

The pilot's voice crackles on too loud...he'd forgotten to compensate for earth-density atmosphere. The standard greeting, followed by the standard looong list of flight experience to put us at east. He addresses the flight attendent by name, and they have a merry chat that is carefully calculated to put us at ease. The crew is happy, and thus we are happy. Sometimes I wonder what else they put that Space Shake to make us so happy.

We're pushed back in our seats at the low-altitude engines of the THSTV kick in. Wow. The power of these engines never fails to impress me. They

are also REALLY LOUD, despite the insulation. The pressure lets up after about a minute, and then we can relax. The newcomers to Space are plastering their faces against the window, watching with wonderment as the sky turns from blue to indigo. It's actually a pretty neat sight, but I feel fine watching it on the television monitors mounted at the front of the cabin. The engines become quieter, though we can still feel the vibration. It's a bit annoying.

Two hours into the flight, we're at Mach 15 over the ocean, hitting the unpowered apogee of the flight. The flight attendent calls attention to this fact. The pilot's voice comes on again, this time sounding a little thin due to the lower-pressure oxygen-nitro gas mix that's kicked in. He explains the separation process, and the monitors illustrate the process with slick computer animation. By the time he's finished, we all feel like we could BUILD a damn shuttle separation mechanism, and we're damn proud, too. We are so wrapped up in the afterglow of this knowledge that we almost miss the shuttle separation. The monitors show us, live, as our shuttle drops away from the main THSTV body...we watch as it pulls clear of us for a few seconds. Then the second-stage burn starts, throwing us once again into the back of our seats. This is a 2G burn that lasts for 30 seconds...and now we're in low earth orbit and in free fall.

The flight attendent unstraps herself and floats around the cabin with practiced ease, checking on the passengers. The monitors are showing the Earth from orbit, and many of the newly-minted space travellers are again mashing their noses against the tiny windows. I hear the kid's voice utter, "It's so wonderful."

We're allowed to unstrap ourselves one-by-one to experience freefall in the forward section of the cabin, where everything (including the monitors) is padded. There are also passengers to kick in the head. The flight attendent, though, has the unenviable job of showing the newcomers how to manuever in space. The kid, he grasps it fairly quickly. One of the older engineers alternates being too jerky and too stiff, and ends up kicking one of the monitors. His horrified expression makes us all laugh and laugh. When it's my turn, I show her my space platform certification. Of course, I am required to perform and tell stories...we are all ambassadors of the Next Frontier.

After about an hour of this, the cabin lights dim. The Pilot informs us that we will be rendezvous with LEO Spaceport *Charleston* in about six hours, so get some rest. Most of the passengers dose off, exhausted by their bout with zero-G. The flight attendent (Renee, or Ren, as she likes to call herself) and I talk for a bit. Her brother worked on the same platform that I had started out on, it turns out. It was one of the early materials processing labs...unfortunately, it had developed problems with its altitude control and burned up over Northern China. Most of her family was involved with the Space Industry now, and planned to help settle one of the outlying lunar outposts. Her brother was already there, working on the L5 station project. I told her about the problems they were having with the mass driver launching system being built at Tycho....some kind of unanticipated background radiation was interfering with the superconductor controls. It's why I'm on-shuttle, in fact. I promise I'll look up her brother and pass along a message. Nice girl, imbued with family values. Maybe the dangerous nature of space

exploration strengthens family ties.

The pilot's voice crackles on once more, announcing the berthing operation with Charleston. Charleston Station is one of the newer American ports, and probably is similar in decor to the compound we had launched from. The monitors show the station approaching. It's a classic power tower configuration, gravity-gradient stable with all the modules on one end. Some of the russian stations are these awful spin-stabilized things that are constantly bobbling in orbit. Only the brilliance of the Russian mathematicians keep them from tumbling into the atmosphere. As we approach the station, I can see that it also boasts a large external fuel tank from a past shuttle mission. It's gigantic...about 100 feet long and 50 feet wide. It's probably being used as a work area or fuel storage, though I wouldn't be surprised if it was being used as scrap for some other project.

As it turns out, the fuel tank is being used as the terminal. It's been fitted with a berthing mechanism that was installed in space, a task that would have been impossible two years ago. Space construction technology continues to advance with amazing speed. The airlock mechanism itself is proudly labeled, "Made in Tycho City, Luna." As we float hand-over-hand through the airlock (yes, it's huge!) the kid reverently passes his fingers over the embossed label. He almost spins out of control in the process, but the flight attendent and I grab him before he can hurt himself. He mumbles a thanks, eyes self-consciously cast towards his belly. The rest of the passengers are all aglow with wonder as they float down the terminal towards the reception area. There is a bank of "Space Phones" that you can use to call home...guess where I'm calling you from, Mom! You can also purchase some postcards ("100% manufactured by post-consumer space waste") and additional feel-good pills from the Space Pharmacy. The tourists are routed to the appropriate zero-G lounge. The people who have come to do work or company-sponsored research meet with their representatives to plan other rendezvous with other destinations in LEO or Luna. You can book flights or hop on as passengers on an OMV through an computerized passage barter.

I wish Ren farewell, and check into the station proper. Security has been stepped up, I notice. Here, it's much more cramped in the Old Style of modular ergonomics. Tycho City maintains a small communications office in module M2-1 for its technical staff that I can use to schedule my flight to Luna on a private shuttle, if I so desired. Most of these shuttles are tubs with low delta-V, taking 2-3 days to reach the Moon, but hey...they're free and they're not too uncomfortable. Plus, there are no children to contend with...a BIG plus if you've ever been trapped in a can with a crying, wheezing little chain-barfing brat who's managed to free itself from its straps. They used to always puke up their space sickness pills before the administration hit upon the whole "Space Shake" idea. Unfortunately, the whole "Space" line has expanded to "Space Jumpers", "Space Phones", "Space Cards"...etc. It's more than slightly nauseating, this blatant commercialization of space.

There's an OTV arriving in just a few hours...according to the manifest, it's one of the old modular ships transshipping a load of hydrogen mined from the earth's upper atmosphere. I'll be able to hitch it to the Moon, using my company barter card to effect payment to the operator. I don't recognize the captain's name...it used to be that we all knew each other. Not anymore.

I head back out to the terminal area to borrow some books. The books are part of an orbiting library system. They are all brought up from earth one by one, many by passengers who donate them to the library. When I'm done with them, I'll give them to someone else on Luna through the library, and eventually they may make their way to L5 when that station is completed. Nothing to do but wait for a while. As usual, my earlier hunger has passed...it will take me a day before my body actually feels like eating.

APPENDIX D

This is the C source code for an X-Object that provides two extensions to Director:

ScrollRect moves a chunk of screen memory.

Copy copies a piece of screen memory into the Macintosh Scrap so it can be pasted through the Clipboard to a Director cast number.

To compile this code, you need the X-Object development kit from Macromedia. Version 1.2 is the latest version. The documentation, however, is still from 1990.

(post-show note: There is a bug in the MethodCopy code. I allocate daPort as a (*GrafPtr), but I do not allocate a GrafPort to use...thus, random memory is written to.)

```
/*****************************************************************
        ScreenUtilities
        For Think C

        Copyright © 1995 by Dave Seah

        Based on Widget XObject Example
        Copyright © 1989, 1990 by MacroMind, Inc.
        All rights reserved.

*****************************************************************/

#include "XObject.h"      // part of XObject Dev Kit

XClassBegin {             /* Due to a peculiarity of the macro, */
        short x1, y1;     /* you must have at least one instance */
        short x2, y2;
} XClassEnd WidgetObj, WidgetObjRec;

/*===============================================================*/
/* - Declarations of methods - */
pascal long MethodNew(MsgId msg, WidgetObj me);
pascal long MethodDispose(MsgId msg, WidgetObj me);
pascal long MethodSetRect(long bottom, long top, long right, long left,
MsgId msg, WidgetObj me);
pascal long MethodScroll(long dy, long dx, MsgId msg, WidgetObj me);
pascal long MethodCopy(MsgId msg, WidgetObj me);

/*===============================================================*/
/*****************************************************************
When creating a code resource, Think C designates the first function
that
appears in the source file that contains main() as the entry point.
If main() is to be the entry point, no other function should precede it.
*****************************************************************/
main()
{

        /* - Message name table - */
        XMessagesBegin
        XMessage "\p- Dave's Screen Utilities XObject v0.2";
        XMessage "\p- build01, 11apr95";
        XMessage "\p-";
        XMessage "\pI      mNew              - Standard creation method";
        XMessage "\pX      mDispose           - Standard dispose method";
        XMessage "\pXIIII mSetRect, l,r,t,b - Set area to scroll";
        XMessage "\pXII    mScroll, dx, dy - scroll by dx and dy";
        XMessage "\pX      mCopy           - copy rect to clipboard";
        XMessagesEnd

        /* - Method dispatch table - */
        XMethodsBegin
                XMethod  MethodNew;
                XMethod  MethodDispose;
                XMethod  MethodSetRect;
                XMethod          MethodScroll;
                XMethod          MethodCopy
        XMethodsEnd

} /* end main() */


/*-------------------------------*/
/* - mNew        Initialize myself - */
pascal long MethodNew(MsgId msg, WidgetObj me)
{
        short er;
        Rect sR;

        XMethodOpen(me);

        SetHandleSize((Handle)me, sizeof(WidgetObjRec));
        /* enlarge to fit */

        er = MemError();

        if(! er)
```

```
        {
                XObjVar(me,x1)=screenBits.bounds.left;
                XObjVar(me,x2)=screenBits.bounds.right;
                XObjVar(me,y1)=screenBits.bounds.top;
                XObjVar(me,y2)=screenBits.bounds.bottom;
        }/* end if */
        else
          SysBeep(1);

        XMethodClose(me);
        return ((long)er);    /* return non-zero if failed to init */

} /* end MethodNew() */

/*-------------------------------------*/
/* - mDispose    Dispose of myself - */
pascal long MethodDispose(MsgId msg, WidgetObj me)
{
        XMethodOpen(me);
        SendSuper(mDispose, me);
        XMethodClose(me);
} /* end MethodDispose() */

/*-------------------------------------*/
/* Scroll an area by amount in dx and dy */
pascal long MethodScroll(long dx, long dy, MsgId msg, WidgetObj me)
{
        RgnHandle updateRgn;
        Rect sR;
        RGBColor color;

        color.red = 0; color.green = 0; color.blue = 0;

        XMethodOpen(me);
                RGBForeColor(&color);


SetRect(&sR,XObjVar(me,x1),XObjVar(me,y1),XObjVar(me,x2),XObjVar(me,y2));
                updateRgn=NewRgn();

                ScrollRect(&sR, (short)dx, (short)dy, updateRgn);
                PaintRgn(updateRgn);

                DisposeRgn(updateRgn);

        XMethodClose(me);

} /* end MethodBeep() */

/*-------------------------------------*/
/* set the region for scrolling */
pascal long MethodSetRect(long x1, long y1, long x2, long y2, MsgId msg,
WidgetObj me)
{

        XMethodOpen(me);
                XObjVar(me,x1)=x1;
                XObjVar(me,y1)=y1;
                XObjVar(me,x2)=x2;
                XObjVar(me,y2)=y2;
        XMethodClose(me);

} /* end MethodCount() */


/*-------------------------------------*/
/* Copy to Clipboard! */

pascal long MethodCopy( MsgId msg, WidgetObj me)
{
        PicHandle myPic;
        GrafPtr * daPort;                    // Handle to a Port!
        OpenCPicParams myCPicParams;
        Rect sR;
        long count;
        short error;
```

```
        XMethodOpen(me);
                GetPort(daPort);

SetRect(&sR,XObjVar(me,x1),XObjVar(me,y1),XObjVar(me,x2),XObjVar(me,y2));
                myCPicParams.srcRect = sR;
                myCPicParams.hRes = 0x00480000;
                myCPicParams.vRes = 0x00480000;
                myCPicParams.version = -2;
                myCPicParams.reserved1 = 0;
                myCPicParams.reserved2 = 0;

                myPic = OpenCPicture(&myCPicParams);

                CopyBits(&(**daPort).portBits,&(**daPort).portBits,&sR,
&sR, srcCopy, NULL);
                if (QDError()) SysBeep(1);
                ClosePicture();

                ZeroScrap();

                count = GetHandleSize((Handle)myPic);

                error=PutScrap(count, 'PICT', (Ptr) *myPic);
                if (error) SysBeep(1);

                KillPicture(myPic);


        XMethodClose(me);
}


#if 0
/*-----------------------------------*/
pascal long MethodTemplate( MsgId msg, WidgetObj me)
{
        XMethodOpen(me);
        XMethodClose(me);
}
#endif



--- end of listing ---
```

POST APPENDIX

I had an observation while making final corrections to this draft: *The appendices end real sudden. It just ends! No parting shot or anything!*

I found this somewhat unnerving. I also felt a lingering guilt over not including **more names** on the dedication page.

I include this section to commemorate the friends and mentors that have helped me break from **engineer-geekdom.** Or maybe they told me it was OK to stay that way.

I wouldn't have made it into the Computer Graphics Design program without the guidance of my chief advisor, **Bob Keough**, when I first visited RIT to see if there was a "computers 'n art" graduate program. Despite my unartistic background, Bob didn't mock my aspirations and encouraged me to submit a portfolio. Thanks Bob!

There was one condition upon entry...take some Graphic Design courses. My great weakness was deemed to be Typography, a summer session taught by **Heinz Klinkon.** It was the first studio art course I had ever taken, and I remember The Terror of being the only person there without a huge ArtBin full of stuff. Though I continued to be terrified for the rest of the course, I gained my first insights into a design *philosophy* from Heinz and his anecdotes. It was my honor to TA summer sessions with him a year later. Today, I see a lot of Heinz's uncompromising but explorative process in my own efforts. Thanks Heinz!

My formative semesters in the Computer Graphics Design program were molded (mashed?) by professor **Jim VerHague**, whose cheerfully-goading attitude eased my initial feelings of dread. He was the role model for my transition into Art, having himself earned advanced degrees in math and science before diving into Design. What would my first year have been without this? Probably much more painful without so much enlightenment! Thanks Jim!

After this kind of introduction, I might have been content to leisurely sit around and learn how to use Photoshop. No such luck with the likes of **Bill Colgrove, Gedeon Maheux,** and **Talos Tsui** setting the pace. They showed me just what it means to **kick ass** through the relentless pursuit of excellence.

Untold thanks goes out to my friends and peers in the Computer Graphics Design program! I wouldn't have had such a great time without the late-working, restaurant-hopping, gossiping, joke-telling, secret-sharing, gift-giving and tip-sharing people that is the Computer Graphics Design program. Never before have I met such a wonderful group of creative individuals.

The very same can be said of my friends in other programs in the College of Imaging Arts and Science, particularly in the Graphic Design, Ceramics, and Film/Video/Photo/Animation departments. My horizons...expanded!

Finally, heartfelt thanks to my dad, **Ingram S. Seah**, who supported my seemingly-bizarre switch from engineering to art without raising a gigantic, asian-style fuss. Thanks imbuing me with a sense of honor, truthfulness and dedication. Thanks for having faith in *me*.